Heriot-Watt University

Masters Thesis

Exploration in Multi-Agent Deep Reinforcement Learning

Author:
Filippos Christianos

Supervisor:
Dr. Stefano Albrecht
Dr. Frank Broz

A thesis submitted in fulfilment of the requirements for the degree of MScR.

in the

School of Mathematical and Computer Sciences

August 2019

EDINBURGH CENTRE FOR ROBOTICS

Declaration of Authorship

I, Filippos CHRISTIANOS, declare that this dissertation is my own original work that is being submitted to Heriot-Watt University, Scotland in partial of the Degree of Master of Science in Robotics and Autonomous Systems. I acknowledge that the original work that is being submitted to Heriot-Watt University has properly been cited and referenced. Some elements of this work may have already been submitted to Heriot-Watt University as part of the dissertation preparatory work under Robotics Research Report (B31AP) and/or Robotics Research Proposal (B31AT). It has not been submitted to any other university or institute of higher learning.

Signed:

Date: 15th August 2019

Abstract

Much emphasis in reinforcement learning research is placed on exploration, ensuring that optimal policies are found. In multi-agent deep reinforcement learning, efficiency in exploration is even more important since the state-action spaces grow beyond our computational capabilities. In this thesis, we motivate and experiment on *coordinating exploration* between agents, to improve efficiency. We propose three novel methods of increasing complexity, which coordinate agents that explore an environment. We demonstrate through experiments that coordinated exploration outperforms non-coordinated variants.

Contents

List of Figures

List of Tables

Abbreviations

BA-MDP	**B**ayes **A**daptive **MDP**
BRL	**B**ayesian **R**einforcement **L**earning
DDPG	**D**eep **D**eterministic **P**olicy **G**radient
DQN	**D**eep **Q** **N**etwork
GP	**G**aussian **P**rocess
IPD	**I**terated **P**risoners **D**ilemma
IQL	**I**ndependent **Q** **L**earning
MADDPG	**M**ulti-**A**gent **DDPG**
MADDPG-AR	**MADDPG** with **A**uto **R**egressive Exploration
MADDPG-TS	**MADDPG** with **T**homson **S**ampling Exploration
MAL	**M**ulti-**A**gent **L**earning
MARL	**M**ulti-**A**gent **R**einforcement **L**earning
MAS	**M**ulti-**A**gent **S**ystems
MDP	**M**arkov **D**ecision **Process**
PO-MDP	**P**artially **O**bservable **MDP**
RL	**R**einforcement **L**earning
TD	**T**emporal **D**ifference

Symbols

s, s'	States
o, o'	Observations
x	Features (deep RL)
a	Action
r	Reward
γ	Discount factor
S	Set of non-terminal states
A	Set of all available actions
$P_a(s, s')$	Transition probability
$R_a(s, s')$	Reward function
$v_\pi(s)$	Value function
$q_\pi(s, a)$	Action-value function
Q	Output of Q network (Q-value)
π	Policy
μ	Deterministic policy
$\pi(a\|s)$	Probability a policy π selecting action a when in state s
G_t	Return following time t
$J_{\boldsymbol{\theta}}$	Performance measure of policy $\pi_{\boldsymbol{\theta}}$

Chapter 1

Introduction

Learning from interactions is presumably the way humans accumulate knowledge of their environment and understand what behaviours are acceptable and rewarding. Reinforcement Learning (RL), a topic widely studied, closely resembles this type of learning; maximising rewards instead of being provided with correct answers or trying to uncover hidden structures in data. Recently, RL has received renewed attention by employing deep networks and solving several hard problems [31, 44, 45, 50].

Multi-agent RL (MARL), attempts to extend learning to environments where multiple agents learn and act simultaneously. The presence of other agents complicates learning but is realistic, and found in many real-world applications such as autonomous driving [27, 42], robotic warehouses [53], and even video games [50]. The abundance of applications makes it a topic of great interest for researchers.

However, MARL faces additional challenges in the form of non-stationarity, multiple equilibria, and the credit assignment problem [8, 33]. In deep RL settings, where value functions are approximated, such problems are exacerbated [24]. Recent research on the topic of deep MARL has made an advance in solving these issues [14, 15, 29, 37].

In this thesis, we focus on an essential aspect of RL, exploration. Exploration is part of an RL algorithm that focuses on gathering information about the environment. By trying new actions in circumstances not encountered before, the agent observes and then learns transitions to new states and rewards. Exploration repeatedly encounters the dilemma of selecting an action that has worked well in the past, versus an action that now seems inferior, but might prove to be better.

With the increasingly large state spaces that deep RL can handle, exploration has an even more critical role. State spaces in robotics, autonomous driving, or other complicated environments, are too large to be thoroughly visited. Therefore, exploration must be efficient and focus on actions that have the potential to be optimal.

As part of our efforts to improve exploration, we propose three novel algorithms for multi-agent environments and demonstrate that exploration dramatically affects performance in deep MARL. Those algorithms operate under the paradigm of *coordinating exploration* during the learning phase but learning a decentralised policy for the execution.

To evaluate our methods, we implement a multi-agent domain: level-based foraging. It is a challenging domain for MARL, requiring agent coordination for successful completion. Agents are required to navigate and operate in a grid environment and are often required to cooperate with others. In this domain, we demonstrate the performance of our methods.

Such coordination of agents could in the future be extended to solve real-world problems. For instance, our interest focuses on robotic warehouses where the tasks, perhaps similarly to level-based foraging, includes coordination for the collection and delivery of items. Currently, warehouses with multiple robots operate under a centralised controller. We hope that research in MARL might provide an alternative, decentralised and thus more scalable, way of planning in such situations.

1.1 Thesis Outline

This thesis consists of six chapters. The second chapter presents a thorough review of the literature in: i. single-agent deep RL (Sec. 2.1), ii. multi-agent deep RL (Sec. 2.2), and iii. exploration through Bayesian RL (Sec. 2.3). Chapter 3 introduces concepts, notations or algorithms that will be used in the methods we propose in Chapter 4. Next, Chapter 5 presents the evaluation of our methods through experiments. Chapter 6 concludes.

Chapter 2

Review of the Literature

2.1 Single Agent Deep Reinforcement Learning

DQN has been shown to be an unstable algorithm, meaning more training does not always equal better performance. It is often the case that the performance of DQN agents is constantly monitored throughout the training, and only the best performing snapshots are kept. This instability is attributed to an overestimation bias introduced with the argmax operator. Double DQN (DDQN) [23] attempts to solve this with a second value network that is used to cross-validate the Q-values. Hasselt et al. [23] show that the overestimation of Q-values is indeed happening in DQN networks and causes the observed instability and the proposed DDQN outperforms them.

Until 2016, deep RL was implemented with the use of an experience replay (Sec. 3.4). This technique was used to counteract the temporal correlation of collected samples but had several drawbacks. By definition, using an experience replay limited deep RL algorithms to off-policy learning. Also, especially when the state representation was an image, a lot of memory and computing power was needed. Mnih et al. [30], with the Asynchronous Advantage Actor-Critic (A3C) algorithm, replace the experience replay by training simultaneously on several parallel threads. The simultaneous training breaks the correlation of the samples and also enables the use of on-policy algorithms.

While typically on-policy algorithms are not as sample efficient as the off-policy equivalent, the advantage of A3C was the capacity to use of all the machine cores (16 in the original paper). This meant that several environments and gradient calculations were running simultaneously, significantly decreasing the real-time training requirements. For the first time, agents solving Atari games were being trained in just a few hours on a typical CPU, without even the need of GPUs.

The above algorithms modelled the policy as $\pi(\cdot|s)$, meaning a probability distribution over the action. Deterministic policies $\mu(s)$ directly output an action when given the state, and follow the policy gradient to improve performance. Deep Deterministic Policy Gradient (DDPG), introduced by Lillicrap et al. [28], combines DQN with the deterministic policy gradients and extends it to *continuous* actions. Continous action space is an interesting extension since it allows for many real-life applications like robotics or autonomous vehicles.

Twin Delayed DDPG (TD3) [16] addresses similar issues to DDQN, such as the overestimation bias. Fujimoto et al. [16], propose the addition of a second value network that learns independently and uses the minimum of the two when formulating the Bellman target. Noticing that similar actions, in continuous action space, should give similar rewards, Fujimoto et al. also add noise to the target action, effectively smoothing the action values. While TD3 appears to be a collection of empirical tricks rather than having a concrete mathematical formulation, it shows improvements in performance over DDPG and is considered state of the art on continuous action environments.

A newer algorithm, Soft Actor-Critic (SAC), was proposed by Haarnoja et al. [22] and is interesting to us since it naturally encompasses exploration.

SAC adds an entropy term to the Actor's training, and the Actor learns to achieve high rewards while also maximising exploration. Haarnoja et al. claims that this is one of the most stable variations of deep RL, achieving the same results with various seeds, while also outperforming the previous state of the art algorithms.

Until now, we listed papers that focused on adapting classical RL rules to deep networks. A common element of off-policy deep RL algorithms is the experience replay that remained unchanged throughout these years. Prioritised Experience Replay (PER) [40] aimed to change that and proposed a priority over experiences which is calculated by the difference of the predicted and the real reward. By prioritising sampling, the network is trained more often with experiences it does not currently predict correctly, effectively decreasing the learning time.

Hindsight Experience Replay (HER) [5] modifies the experience replay to tackle a different problem. As sparse rewards are a challenge in RL, HER modifies experiences and its rewards to achieve an implicit curriculum and learn in these settings. In essence, the goal of each episode is modified to match what was randomly achieved, and the agent instead of having nothing to learn learns those modified goals. Learning with such binary and sparse rewards is very important in RL since it avoids the need to hand-engineer reward structures.

2.2 Multi-agent Deep Reinforcement Learning

In the previous section, we have described the current state of the art in single-agent deep RL. The introduction of several agents adds a significant challenge to RL because it introduces non-stationarity [33]. There are many attempts to adapt deep RL algorithms to work in such settings, with various degrees of success. Fortunately, even while the Markov assumption breaks in multi-agent settings (Sec. 3.3), RL algorithms still work sufficiently well.

Starting with a change to the experience replay, Foerster et al. [15] deal with the non-stationarity and the instability that it introduces. Specifically, since the critical factor to the instability is the constantly changing behaviour of the opponents, Foerster et al. propose two methods that address it. Both methods, marking data as obsolete as time progresses and adding a unique fingerprint to collected experience, perform equally better when combined with Independent DQN (IDQN).

Shortly after attempting to stabilise training with changes in the experience replay, Foerster et al. [14] introduce the COMA algorithm. COMA adheres to the paradigm of *centralised* learning - *decentralised* execution: agents that are trained in a lab or simulated environment but then can act autonomously. To achieve this, COMA uses a single and centralised Critic which takes as an input the full state and produces Q-values for all the joint actions. The decentralised Actor uses a counterfactual baseline that marginalises the agent's actions and keeps the other Q-values fixed.

QMix [37] is a newer algorithm from the same group, that relaxes some of the hard constrains of COMA. QMix is a method that lies between IDQN and COMA and uses Q-networks for each agent but then *mixes* them monotonically using another network to generate a Q^{total} network. The centralisation of the Critic these methods proves to be very important and provides a stable target for the actors.

Finally, MADDPG (detailed in Chapter 3) was proposed by Lowe et al. [29]. The algorithm combines DDPG with tricks that improve learning in a multi-agent environment. The paper suggests the use of a centralised Critic, with inputs of all actions and observations which reduces the variance of Q-values. Also, the novelty of the paper lies to the proposed policy *ensembles*, where several policies are trained for each agent and sampled during training to minimise the non-stationarity problem. However, it is unclear how each component affects the performance of the algorithm, and as discussed in the previous chapter, we have found MADDPG to work equally well without the use of policy ensembles. In any case, it is clear that MADDPG outperforms the independent version of DDPG and is considered state of the art in multi-agent systems.

2.3 Exploration through Bayesian Reinforcement Learning

This section focuses on the literature concerned with Bayesian Reinforcement Learning (BRL) methods. In the previous sections, we discussed deep RL methods, but we did not mention exploration. Often, deep RL makes use of simple exploration methods. Bayesian RL, by modelling uncertainty, focuses on *efficient exploration.* As of now, there is not much work in combining Bayesian methods with deep networks (except for the work by Gal and Ghahramani [17] which we discuss at the end of this section), and we either focus on tabular settings or briefly cover model-free methods based on Gaussian Processes (GPs).

2.3.1 Model-Based BRL: Using Bayes-Adaptive MDPs

Model-based, as the name implies, consists of algorithms that create and maintain a model of the environment. In BRL literature, we find extensions of Markov Decision Processes (MDPs, background on MDPs can be found in Sec. 3.1.1) which unfortunately are often intractable. As such, we discuss methods which are meant to approximate solutions either online or offline.

Bayes-Adaptive MDPs (BA-MDPs) by Duff and Barto [11], is a model-based approach to optimal exploration in MDPs. Specifically, the BA-MDP framework is designed to allow agents to reason about their uncertainty of the underlying MDP dynamics. Formally, a BA-MDP is defined similarly to an MDP but replaces the states with a set of hyper-states, which represent *all* the possible transitions into the future. The respective transition function models the belief of transitioning to each of the hyper-states and is often populated with Dirichlet posteriors [11, 21].

It is important to note that in BA-MDPs, the hyper-states grow exponentially with the horizon t, and on a fully connected underlying MDP, we can expect $|S|^t$ states. Thus, the computational challenges are apparent and actual solutions are tractable only within simple environments.

Computational challenges aside, this representation of MDPs offers remarkable advantages, and most notably an optimal solution to the exploration vs exploitation dilemma.For instance, using the Bellman equation and a t-step horizon, we can derive the optimal value function, which if maximized with a policy results in a *Bayes-optimal* policy. Bayesian inference is thus used to select actions in an optimal way, given the existing uncertainty. Many of the following works are trying to approximate this value function.

An algorithm that paved the way of Bayesian RL is one that builds on Q-Learning and was proposed by Dearden et al. [10] in the '90s. In Bayesian Q-Learning, the actions are selected by calculating the expected value of information on top of the expected reward. This approach does not solve the BA-MDP, but instead, first samples MDP from posterior Dirichlet posteriors and then solves them to create a distribution on the Q-values. Then Dearden et al. define the *value of perfect information* (VPI) for a state-action pair in a myopic manner (considering only the immediate transition). As such, action selection is made by maximising the sum of the VPI with the Q-value.

The *Beetle* algorithm that is introduced by Poupart et al. [36] in 2006, gathers a sample of hyper-states through continuous simulations of the BA-MDP. Then, point-based backups are done, essentially solving a Partially-Observable MDP (POMDP, Sec. 3.1.1.1) where the belief states are replaced by the hyper-states. This algorithm faces deteriorating computational issues with an increasing horizon since the number of terms in the resulting polynomials increases exponentially. With the use of basis functions, and bother techniques, the issue above is mitigated. Experiments in small domains show *Beetle* to work well, but for implementation in more complex systems, prior knowledge of the domains should be used to limit the parameters.

Bayes-Adaptive MDPs can also be extended to Bayes-Adaptive POMDPs (BA-POMDPs) as shown by Ross et al. [38]. In this work, the authors propose this extended framework in order to tackle the domains where the state is a hidden variable. This model helps the planning of optimal actions which aim to concurrently identify the current state, explore the underlying model and exploit the knowledge to acquire rewards. The transition and observation probabilities in a BA-POMDP are considered unknown, and their uncertainty is modelled with Dirichlet distributions. Notably, the belief and optimal value functions are still similar to POMDPs but need to be computed in the new *extended* state space. Understandably, exact solutions over both long horizons and large state spaces are intractable and approximate solutions are proposed.

Ross and Pineau [39] address the scalability of online methods in larger domains. They argue that there are two main reasons which restrict BRL methods to small domains. The first is that they need a large amount of data to make a usable model, something which they tackle by using factored representations of the dynamics of a system. This leads to fewer parameters and is done with the use of dynamic Bayesian networks. The second reason is that planning in Bayesian RL seems intractable due to the full posterior space which should be taken into consideration (see 2.3). The authors in this work propose the use of an online Monte Carlo algorithm, an approximating technique which also since it is online only has to start from the current posterior. Nevertheless, the dependency on tree structures hinders the ability to generalise and extend to the continuous space and has issues with online planning, as discussed above.

More recently, Katt et al. [26] propose a Factored Bayes-Adaptive POMDP extension (FBA-POMDP). The factorisation is possible in POMDPs due to the conditional independence between the variables. As such, it is possible to represent the dynamics with the reduction of parameter space more efficiently. The authors show that the factorisation can also be similarly applied in BA-POMDPs. Finally, they introduce a solution method based on Monte Carlo Search Tree, which has convergence guarantees and outperforms state of the art approaches.

Bayes-Adaptive methods such as the ones described show that principled solutions to the exploration vs exploitation trade-off in MDPs exist. While the state space is countably infinite, near-optimal solutions are viable and could potentially be extended to solve more complex domains.

2.3.2 Model-Free BRL

Model-free, in contrast to Model-based approaches, do not directly model the environment and thus do not involve planning. Instead, they learn by trial and error, observing the results of their actions for each state. Model-free BRL usually incorporates Gaussian Processes (GPs), which naturally express the posteriors along with the uncertainty.

Unfortunately, modern RL needs to use vast amounts of collected data (for example, in environments such as Atari games), and GPs currently have very high computational requirements to process this data. As such, the use of GPs in RL has not seen much focus. We briefly outline some of the most advanced methods.

Gaussian Process Temporal Difference (GPTD), introduced by Engel et al. [12], is a Bayesian approach for value function estimation in domains with continuous state spaces. They first define a generative model for the value function using a Gaussian prior and derive a posterior distribution. An online sparsification is used, where the algorithm drops samples if they are not required to maintain an accuracy threshold[1]. This leads to significantly fewer samples and thus decreased computational requirements. As the experiments show, it manages to produce not only a value function of a continuous-space maze but a value variance as well.

In a later work, Engel et al. [13] address the first two issues by modifying the initial model and incorporating a discounted return process into the value function. They also offer a SARSA extension to GPs, GPSARSA. The proposed algorithm directly derives from the idea of extending TD to estimate the Q values. Finally, the model-free methods discussed until now seemed to ultimately lead towards an actor-critic implementation of BRL [12, 13, 20]. Ghavamzadeh and Engel [19] made the first such attempt by replacing parametric critics (e.g. [30]) using a GP for the critic.

The use of GPs for estimating uncertainty in RL is certainly appealing and can offer theoretical guarantees; however, the computing requirements render it unrealistic. Gal and Ghahramani [17] make use of dropout on neural networks as a Bayesian approximation. Dropout has been extensively used empirically in deep learning, especially in domains such as vision in order to combat overfitting. However, Gal and Ghahramani show the link between dropout and Bayes mathematically. Interestingly, while they mostly focus on vision, they also show uses of dropout in single-agent RL settings using DQN. In Sec. 4.2 we will discuss how to apply dropout in *multi-agent* settings, in an Actor-Critic architecture.

[1]Even on these simpler environments, processing all the data is computationally hard, further reinforcing the view that implementations in large state-space domains are currently intractable.

Chapter 3

Technical Background

This chapter introduces the technical background that is used in the rest of this thesis. Section 3.1, Tabular RL, is useful for introducing notation, as well as basic RL concepts such as common algorithms. Next, Section 3.2, Deep RL, presents the extension of RL to deep network settings. Finally, multi-agent RL in Section 3.3, introduces the changes to RL when multiple agents co-exist in an environment. Throughout this work, we keep consistent with the notation of the majority of the RL literature and with Sutton and Barto [47].

3.1 Tabular Approaches to Reinforcement Learning

RL is the part of machine learning that is concerned by agents learning through acting on an environment and observing the results of their actions. RL is not considered supervised or unsupervised learning and instead forms the third branch of machine learning on its own [47]. RL differs from unsupervised learning since it does not try to uncover hidden structure in the data but instead tries to maximise a reward signal. Also, in contrast to supervised learning, the agent does not receive correct examples of behaviours.

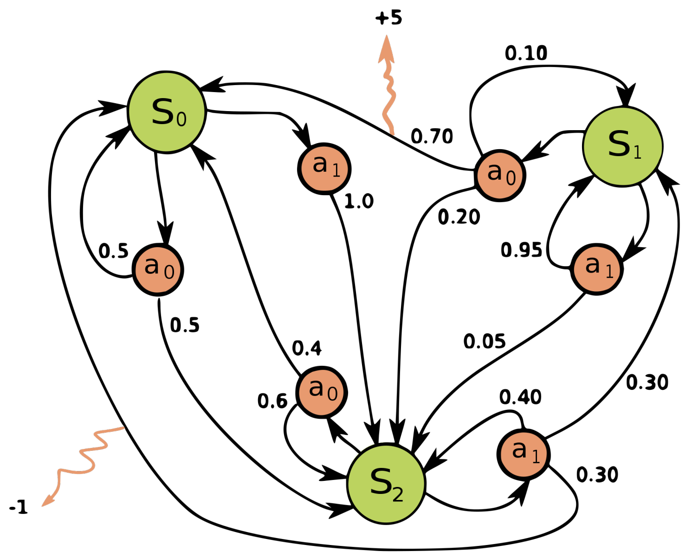


FIGURE 3.1: An example MDP with three states and two actions. The transition probabilities to each state after taking an action from a state are also displayed. Successfully transitioning to S_0 from S_1 and S_2 rewards 5 and -1 respectively.
∘ *source: waldoalvarez* [CC BY-SA 4.0]

In essence, RL consists of an agent acting in an environment, optionally creating a model by learning the underlying functions, with the goal of learning to maximise the rewards of its actions (or learning a *policy*). Next, we formulate the problem with the help of a mathematical framework, Markov Decision Processes and present RL algorithms that learn under those settings.

3.1.1 Markov Decision Processes

Markov Decision Processes (MDPs), an extension of Markov Chains, is a mathematical framework for modelling sequential decision problems that can be solved using RL. MDPs can encapsulate the process of an agent observing the state of an environment, acting on it, receiving a reward, and finally stochastically reaching a new state.

An MDP is defined by the tuple $\langle S, A, P_a, R_a \rangle$ where S is the set of states, A is the set of actions available for each state, $P_a(s, s')$ is the probability of transitioning to state s' from s by taking action a. Finally $R_a(s, s') \rightarrow \mathbb{R}$ is a reward function giving the immediate *reward* or *reinforcement* received under a transition.

The *Markov property*, which is satisfied by MDPs, requires future states of the process to only depend on the current state-action tuple $\langle s, a \rangle$. Specifically, it describes a system where the information of the current state can capture its entire history. The Markov property is an assumption used in MDPs[1] in order to model the particular problems [21, 47].

Solving an MDP means finding the optimal policy π^*, which will yield the maximum reward for an agent. A stochastic policy π matches states to the probability of selecting actions. Specifically, $\pi(a|s)$ is the probability of selecting action a when the MDP is in state s and should be defined $\forall a \in A$ and $\forall s \in S$. This optimal policy can be found using Dynamic Programming when the transition and reward probabilities are known. When those are unknown, then the problem becomes one of RL [47], where model-based approaches attempt to learn P_a and R_a, and model-free directly search for a near-optimal policy.

In this thesis, we will also refer to a *deterministic* policy $\mu(s)$ (sometimes refered to in the literature simply as $\pi(s)$) which maps a state to a deterministic action. Using a different symbol μ clearly differentiates between the stochastic and the deterministic policies. Naturally, a deterministic policy can also be written as a stochastic one such that $\forall s_i \in S \; \exists \; a_j \in A \colon \pi(a_j|s_i) = 1$.

We define the *value function* of a state under a policy as $\upsilon_\pi(s)$ as such:

$$\upsilon_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s \right], \forall s \in S \tag{3.1}$$

[1]As we examine later, this assumption often breaks in multi-agent systems, where the entire interaction history is needed to model opponent behaviours.

where $\gamma \in [0, 1]$ is the discount factor, which weights results in the future differently, and r_t is the reward received in timestep t. The *value function* can be explained as the expected rewards when the agent is in a state s and will follow the policy π. Similarly, we define the value of taking an action a while in s, $q_\pi(s, a)$ as

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right] \tag{3.2}$$

called the *action value function*, and denoting the expected reward when an action is taken and then following the policy π. Due to the Markov property, we can expand Eq. 3.1 recursively:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S \end{aligned} \tag{3.3}$$

Eq. 3.3 is called the *Bellman Equation*, which again expresses the value of a state by connecting it to future states. It is a sum over all the actions, future states, and rewards, weighted by their respective probabilities.

Finally, we define as *return* the sum of rewards following time t as

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

3.1.1.1 Partially Observable MDPs

A well-studied generalisation of MDPs is the Partially Observable MDP or POMDP. In this generalisation, the MDP controls the underlying system, but the agent in the environment cannot fully observe the state of the system. Hence we have the addition of the set of observations Ω, and the probability

$O(o|s', a)$ of observing o when arriving in the state s' having taken the action a. The resulting tuple $\langle S, A, \Omega, P_a, O, R_a \rangle$ defines this model.

POMDPs are often used when modelling real-life problems since they encompass the element of uncertainty in perception. Since the agent does not directly observe the state of the environment, it must hold a belief for the current state, which is updated while the agent traverses the environment. As such, we define as *belief state*, a distribution over latent states. Consequently, the uncertainty of the true state creates optimal actions that are meant to gather information that can be later exploited [21]. Finally, policies are mappings from belief states b to actions: $\pi(a|b)$.

Nevertheless, as shown by Kaelbling et al. [25], solving POMDPs is often intractable when there are more than a few observations, states, and actions. Thus, approximate solutions emerged [34, 35, 46] to tackle larger domains. Those solutions do come with the trade-off of requiring a full model of the POMDP, which is often a strong assumption.

3.1.2 Temporal Difference

Many RL algorithms have been developed throughout the years, solving MDPs either using Dynamic Programming or with Monte Carlo ideas. However, it can be argued that neither managed to influence RL as much as Temporal Difference (TD). TD methods combine Dynamic Programming and update using previously learned estimates, with Monte Carlo, which learns through direct interaction with the environment.

Q-Learning, a prime example of TD learning and developed in the late '80s by Watkins [51], learns the action-value function in a tabular manner. Watkins manages to combine optimal control and TD methods to one of the most studied RL algorithms. A Q-Learning agent interacts with the environment and continuously updates the action-value table using

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)) \tag{3.4}$$

where α is the learning rate and γ the discount factor. Equipped with this table, an agent can maximise its rewards by greedily selecting actions, effectively learning a policy.

Learning a different policy than the one being executed to collect samples is referred to as *off-policy* learning. The advantage of off-policy is that the agent can directly learn the optimal policy. *On-policy* algorithms work, as the name suggests, by improving the policy that is being executed offering different advantages such as more steady improvement. The specific differences between off and on policy variants are often too subtle and environment-dependent to discuss in this thesis.

Another well known is algorithm is SARSA [47, p. 129], which resembles Q-learning but replaces the max operator with an on-policy equivalent as such:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1})) \tag{3.5}$$

The action a_{t+1} is the action that will be performed next under the current policy: which makes it an on-policy algorithm.

3.1.3 Exploration

An important note here is that algorithms (e.g. Q-Learning) should not interact with the environment exclusively with a greedy policy. Doing so, would potentially lead to repeated execution of the same actions, and often never seeing at all some transitions. In RL, collecting a variety of interactions is called *exploration* and is critical to learning. Often, exploration is achieved

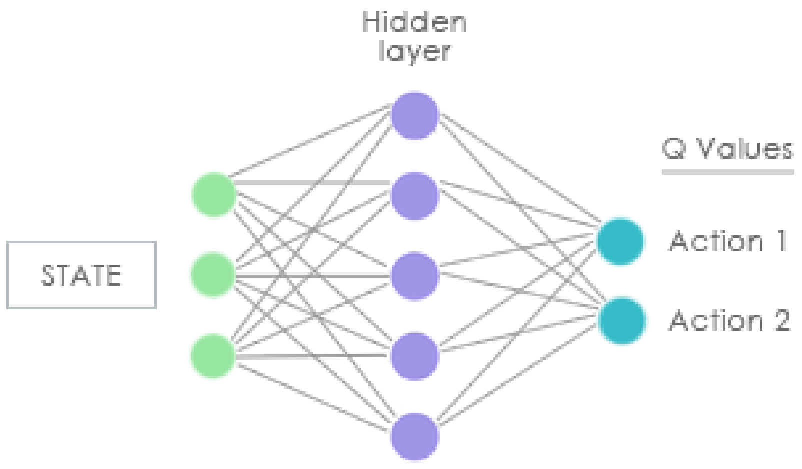


FIGURE 3.2: A DQN network. Takes as an input the state and outputs Q-values for *all* possible actions

by adding randomness to the action selection, ensuring that all state-action transitions will be eventually visited.

The simplest exploration method is called ϵ-greedy. A *small* ϵ is set, and an agent following an ϵ-greedy policy performs the greedy action with probability $1 - \epsilon$ or a uniformly random action with probability ϵ.

Another, more efficient exploration method that we also use later in Chapter 4, is Thomson Sampling. Thomson sampling is a heuristic for addressing the exploration-exploitation dilemma and has been used primarily in the multi-armed bandit problem. A Bayesian posterior $P(\theta \mid \mathcal{D})$ on the parameters θ of the model is held, while the domain is being explored and observations $\mathcal{D}$ are gathered. Then the algorithm samples the parameters of a model $theta'$ from the posterior and chooses an action that maximises the expectation of the reward given the drawn parameters $\max_{a'} \mathbb{E}(r \mid a', \theta')$.

3.2 Deep Reinforcement Learning

The tabular implementations of those algorithms do not scale to high dimensions and require either simple domains or a high-level representation of the state. Difficulty in scaling is the result of requiring to visit and

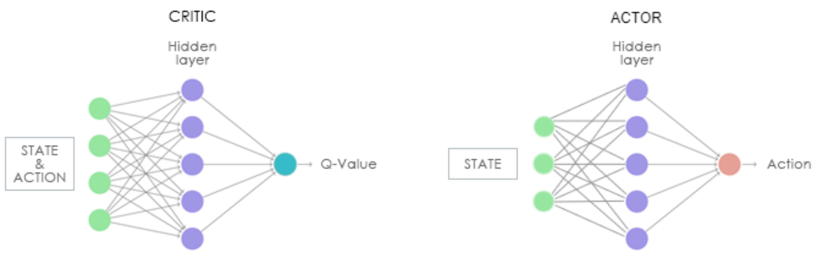


Figure 3.3: The Actor-Critic architecture. Two networks are used, the actor (or policy) network for selecting actions, and the critic (or value) network for training the policy.

calculate the update for all individual states. In this section, we will describe algorithms that utilise deep Neural Networks to approximate Q-values and generalise them to similar states.

The first and most impactful example of such an algorithm is DQN [31], which was the first deep RL algorithm to solve Atari games. Specifically, DQN, also seen in Alg. 1, uses a network architecture that has the state as an input and the Q values of all actions as an output (as seen in Fig. 3.2). Therefore, the loss function, defined as

$$L = \frac{1}{2}(Q(s,a) - (r + \gamma \max_{a'} Q(s',a')))^2 \tag{3.6}$$

can be minimised using a gradient optimiser over the parameters of the network to train a network that can approximate Q-values.

Another addition to the tabular Q-learning is the experience replay, also called replay buffer in the literature, denoted as D. During training, all transitions are stored in a data structure, and mini-batches are drawn for backpropagation. This technique breaks the temporal correlation between samples and allows for more stable overall improvement. Finally, Mnih et al. [31] add another network $\hat{Q}$, the *target* network, that is updated to slowly follow the Q network and acts as a more stable target.

Actor-Critic is another architecture which uses two networks (see Fig. 3.3). The 'Critic' estimates the value function as discussed above, and in turn, the 'Actor' updates the policy as suggested by the value network. Actor-Critic

Algorithm 1 DQN Algorithm

```
Require: Initialize replay memory $D$ with capacity $N$
Require: Initialize action-value network with random weights $\theta$
Require: Initialize target network with weights $\theta^- = \theta$

 1: function $\epsilon$-GREEDY(Q)
 2:     with probability $\epsilon$ select random action $a_t$
 3:     otherwise select $a_t = \text{argmax}_a Q(s, a; \theta)$
 4:     return $a_t$

 5: for timestep $t$ in every episode $1 \dots M$ do
 6:     $a_t \leftarrow \epsilon$-greedy (Q)
 7:     Execute $a_t$ and observe $s'$ $r$
 8:     Store $(s, a_t, s', r)$ in $D$
 9:     Sample mini-batch $(s_j, a_j, s'_j, r_j)$
10:     $y_j \leftarrow \begin{cases} r_j, & \text{if} s'_j \text{is terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(s'_j, a'; \theta^-), & \text{otherwise} \end{cases}$
11:     loss $\leftarrow (y_j - Q(s_j, a_j; \theta))^2$
12:     Gradient descend step on loss with respect to $\theta$
13:     Every $C$ steps $\theta^- \leftarrow \theta$
```

methods have been shown to be more stable and perform considerably better [30].

A well known Actor-Critic algorithm is Deep Deterministic Policy Gradient (DDPG) [28]. This algorithm combines DQN and the tabular DPG and learns a deterministic policy μ. The update rule of the Actor stems from the deterministic policy gradient theorem, which solves the gradients of the policy with respect to the value function. The algorithm, Alg. 2, is very successful with continuous action states in deep networks. Later, we show the multi-agent extension of this algorithm, called MADDPG.

3.3 Multi-agent Reinforcement Learning

Multi-agent Learning (MAL) concerns itself with domains where multiple agents learn at the same time, potentially creating ever-changing behaviours while agents try to adapt to each other. MAL, in contrast to single-agent

Algorithm 2 DDPG Algorithm

```
Initialize replay memory $D$ with capacity $N$
Initialize policy parameters $\theta$ and Q-function parameters $\phi$
Initialize target network with weights $\hat{\theta} = \theta$ and $\hat{\phi} = \phi$
Initialize $0 < \tau < 1$ as the target update hyperparameter

1: for timestep $t$ in every episode $1 \dots M$ do
2:     $a_t \leftarrow \text{clip}(\mu_\theta(s) + \epsilon, a_{min}, a_{max})$, where $\epsilon \sim \mathcal{N}$
3:     Execute $a_t$ and observe $s'$, $r$
4:     Store $(s, a_t, s', r)$ in $D$
5:     if $t \mod T_{update} = 0$ then
6:         Sample mini-batch $B$: $(s_j, a_j, s'_j, r_j)$ from $D$
7:         $y_j \leftarrow \begin{cases} r_j, & \text{if} s'_j \text{is terminal} \\ r_j + \gamma Q(s'_j, \mu_{\hat{\theta}}(s'_j); \hat{\phi}), & \text{otherwise} \end{cases}$
8:         $Q_{gradients} \leftarrow \nabla_\phi \frac{1}{|B|} \sum_B (Q_\phi - y_j)^2$
9:         Gradient descend step on $Q_{gradients}$ with respect to $\phi$
10:        $\mu_{gradients} \leftarrow -\nabla_\theta \frac{1}{|B|} \sum_B Q_\phi(s, \mu_\theta(s))$
11:        Gradient descend step on $\mu_{gradients}$ with respect to $\theta$
12:        $\hat{\phi} \leftarrow \tau\hat{\phi} + (1-\tau)\phi$
13:        $\hat{\theta} \leftarrow \tau\hat{\theta} + (1-\tau)\theta$
```

learning, needs to work in environments where the dynamics change, often called *non-stationary*. The literature of MAL usually revolves around Markov Games, focusing mostly (but not exclusively) in zero-sum or repeated games such as rock-paper-scissors or prisoners dilemma [43]. Like the rest of RL, we can distinguish model-free and model-learning approaches.

Formally multi-agent systems can be represented as Markov games, an extension of POMDPs that may include more than one agent. The definition of a Markov game for N agents contains sets of actions $A_1, \dots, A_N$ and sets of observations $O_1, \dots, O_N$. The observations are a partial description of the full state $s \in S$ from the perspective of each agent. The transition function $S \times A_1 \times \dots \times A_N \mapsto S$ takes as input the state and the actions of all agents and maps to a state. Finally, each agent aims to maximize the discounted sum $\sum_t \gamma^t r_i^t$ of the individual rewards r_i.

Model-based MAL tries to create a model of the opponent behaviour [4], compute and choose the best response, and after observing the reaction, update its model. This iteration was first introduced as *fictitious play* by

Brown [7], and is now the basis of many model-based methods. Model-free, by not creating models of opponents, focuses on the agent's actions and often ignores others entirely [41].

MAL faces many challenges [43], one of the most significant ones being the non-stationarity [33]. Specifically, each agent updates its policy individually, throwing off other agent's policies, creating a vicious circle of moving targets. Another major challenge is the credit assignment problem, being the inability of agents to understand if they were rewarded for their actions or other agent's. As such, learning in such settings is more complicated and is studied separately.

In recent years, deep multi-agent RL algorithms have been proposed to tackle such environments, which we overview in Sec. 2.2. An algorithm that stands out, and will be used in our thesis is Multi-Agent DDPG (MADDPG), proposed in 2017 by Lowe et al. [29]. MADDPG is an extension of DDPG to the multi-agent domain and is explicitly redesigned for domains requiring agent coordination. The main difference between several independent agents using DDPG is the use of centralised Critics, learning on the joint observations and actions, and is trained by minimising the loss

$$\begin{gathered}\mathcal{L}\left(\theta_i\right)=\mathbb{E}_{\boldsymbol{o},a,r,\boldsymbol{o}'}\left[\left(Q_i^{\mu}\left(\boldsymbol{o},a_1,\ldots,a_n\right)-y\right)^2\right], \\ y=r_i+\gamma Q_i^{\mu'}\left(\boldsymbol{o}',a_1',\ldots,a_n'\right)\big|_{a_j'=\boldsymbol{\mu}_j'\left(o_j'\right)}\end{gathered} \tag{3.7}$$

The actors however remain decentralised in order to run independently at execution time and are updated using

$$\nabla_{\theta_i}J\left(\boldsymbol{\mu}_i\right)=\mathbb{E}_{\mathbf{x},a\sim\mathcal{D}}\left[\nabla_{\theta_i}\boldsymbol{\mu}_i\left(a_i|o_i\right)\nabla_{a_i}Q_i^{\boldsymbol{\mu}}\left(\mathbf{x},a_1,\ldots,a_n\right)\big|_{a_i=\boldsymbol{\mu}_i(o_i)}\right] \tag{3.8}$$

The original MADDPG work also proposes training several policies for each agent, called policy ensembles. However, we empirically found that this does not improve performance, and the main contribution is the use of a centralised Critic. Another important aspect is that MADDPG has been shown to perform well in discrete action spaces (in contrast to DDPG), with the use of Gumbel-Softmax sampling. Specifically, this distribution is sampled, which helps exploration, and the actions are converted to discrete with one-hot encoding. When we refer to MADDPG from now on, we referred to this simplified variation, also seen in Alg. 3.

Algorithm 3 MADDPG Algorithm

```
    Initialize replay memory $D$ with capacity $N$
    Initialize policy $\theta_{1\ldots n}$ and Q-function parameters $\phi_{1\ldots n}$ for $n$ agents
    Initialize target network with weights $\hat{\theta}_i = \theta_i$ and $\hat{\phi}_i = \phi_i$ for $i = 1 \ldots n$
    Initialize $0 < \tau < 1$ as the target update hyperparameter

 1: for timestep $t$ in every episode $1 \ldots M$ do
 2:     $\boldsymbol{a} \leftarrow \{\}$
 3:     for $i$ in $\{1 \ldots n\}$ do
 4:         Append $a_i = GumbelSoftmax(\mu_{\theta_i})$ to $\boldsymbol{a}$
 5:     Execute $\boldsymbol{a}$ and observe $\boldsymbol{o}$, $\boldsymbol{o'}$, $\boldsymbol{r}$ of all agents
 6:     Store $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{o'}, \boldsymbol{r})$ in buffer
 7:     if $t \mod T_{update} \neq 0$ then
 8:         Continue
 9:     for $i$ in $\{1 \ldots n\}$ do
10:         Sample mini-batch $B$: $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{o'}, \boldsymbol{r})$ from $D$
11:         $y \leftarrow \begin{cases} r_i, & \text{if } o'_i \text{ is terminal} \\ r_i + \gamma Q(\boldsymbol{o'}, \boldsymbol{a}; \hat{\phi}_i), & \text{otherwise} \end{cases}$
12:         $Q_{gradients} \leftarrow \nabla_{\phi_i} \frac{1}{|B|} \sum_B (Q(\boldsymbol{o}, \boldsymbol{a}; \phi_i) - y)^2$
13:         Gradient descend step on $Q_{gradients}$ with respect to $\phi_i$
14:         $\mu_{gradients} \leftarrow -\frac{1}{|B|} \sum_B \nabla_{a_i} Q(\boldsymbol{o}, a_1 \ldots \alpha_n; \phi_i) \nabla_{\theta_i} \mu(o_i; \theta_i)$
15:         Gradient descend step on $\mu_{gradients}$ with respect to $\theta_i$
16:         $\hat{\phi}_i \leftarrow \tau \hat{\phi}_i + (1 - \tau) \phi_i$
17:         $\hat{\theta}_i \leftarrow \tau \hat{\theta}_i + (1 - \tau) \theta_i$
```

3.4 A Short Summary of the Terminology

For the convenience of the reader, we have summarised the most important terminology that will be repeatedly used from now on (especially Chapter 4).

- Value Network: also called Critic in Actor-Critic algorithms, is the network that learns to approximate the state-action value $Q(s, a)$.
- Policy Network: found in Actor-Critic algorithms, directly outputs the best action given a state ($\pi(s)$ or $\mu(s)$ in DDPG).
- Target Networks: are copies of the main networks, which are slowly updated for example by periodically copying the full parameters. They are meant to offer a stable target when training the main networks.
- On/Off policy: are two categories of algorithms in RL. On-policy algorithm directly train the network they use to collect data, while off-policy may train on data collected in a different manner. Algorithms using an experience replay are always off-policy.
- Exploration: defines the method on which we collect data when interacting with the environment. Usually includes stochasticity, in order to maximise the visited state space.
- Experience Replay: also known as Replay Buffer, is a data structure that stores all the experiences collected by the agent. Batches of experience are sampled and used for training, and helps break the time-correlation of samples.
- Non-stationarity: often found in multi-agent RL, and also referred to as the moving target, is the attribute of an environment that makes the optimal policy change over time. In multi-agent RL it is the main cause of instability, since when an agents behaviour changes, the policy of the other agents must be adapted.

Chapter 4

Methods

An observation that underpins the new exploration methods proposed in this thesis is that while, in multi-agent systems, policies should be eventually decentralised, the exploration can benefit from centralisation. Centralising and coordinating the exploration, as discussed below, generates a richer experience replay, with more instances of valuable cooperation.

In this chapter, we propose three distinct methods of exploration. The first, *centralised ϵ-greedy* (Sec.4.1), is a novel variation of the commonly used exploration technique ϵ-greedy. Next, we adapt the approximate Bayesian exploration using dropout from Gal and Ghahramani [17] to work in multi-agent Actor-Critic settings (Sec. 4.2). Finally, we introduce a novel exploration method inspired by auto-regressive models that focus on agent coordination during exploration (Sec. 4.3).

4.1 Centralising Exploration

The centralisation of exploration follows the centralised learning and decentralised execution paradigm, commonly used in multi-agent learning [14, 29, 37]. During centralised learning, often in lab conditions or a simulation, agents share knowledge or observations which helps stabilise the training of

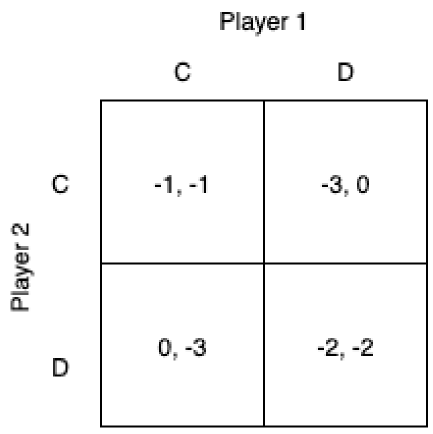

Player 1
C
D
Player 2
C
D
-1, -1
-3, 0
0, -3
-2, -2

FIGURE 4.1: The prisoners dilemma where players 1 and 2 can each perform the actions (C)ooperate and (D)efect. This is the table of rewards where players 1 and 2 receive their reward respectively. Illustrated are the probabilities of a joint action if the greedy action for both agents is calculated to be defect.

a decentralised policy. Later, this policy is used in decentralised execution, where this knowledge is no longer public. Despite the prevalence of sharing observations and other knowledge during training —mostly used for improving the critics or value function approximators— the use of that knowledge for *efficient* exploration has not been studied. We propose *centralised exploration*, a mechanism that coordinates the exploration while having access to the observations and rewards of all agents.

To introduce the intuition behind centralised exploration, we study the classic game theory game *Prisoner's Dilemma*. The game consists of two criminals, the players, apprehended for a heist. However, the prosecutor does not have sufficient evidence to pursue the full charges. He is offering the opportunity to the prisoners to betray each other and lessen their punishment. Thus, each player is presented with two actions: accept the opportunity and *defect* or stay silent and *cooperate* with the other. This leads to three distinct scenarios: both staying silent (C/C), and be imprisoned for lesser charges; both betraying each other (D/D) and serving time for the full crime minus the deal; and finally, one betraying the other, and set free, while the other serves extended prison time (C/D or D/C). Fig. 4.1 also shows the rewards for each player.

In prisoners dilemma, while both agents can benefit from cooperating, a rational agent will always choose to defect. The reason for this is that defection always leads to higher payoff than cooperation regardless of the opponent's choice. Hence, mutual defection is the only *Nash Equilibrium* in this game.

An extended version of this game is the *Iterated Prisoner's Dilemma* (IPD) where the game is repeated a known, unknown, or infinite number of times. In this game, the agents have the opportunity to penalise each other or learn better strategies. IPD has been widely studied in the game theory community, and strategies, including Tit-For-Tat [6] have been shown to work well. Despite this, it is tempting to apply the Bellman Equation (Eq. 3.3) in the manner of Independent Q-Learning (IQL) to study ϵ-greedy exploration in this simple domain.

In this setting, two independent agents following ϵ-greedy run the risk of repeatedly punishing each other during exploration and never learning to cooperate. In fact, higher ϵ leads to worse chances of converging to the optimal C/C strategy, as experimentally shown in Sec. 5.1. In addition, independently exploring leads to many visits to the C/D or D/C actions which are punishing for one of the agents.

A solution to this conundrum comes naturally in the form of centralised ϵ-greedy. Specifically, in this variation, ϵ is used to define a probability of *all* agents exploring instead of each agent having its own, independent probability. Formally, with probability $1 - \epsilon$, all agents will choose their greedy actions, and else choose uniformly randomly. More details can be seen in Alg. 4.

An example that demonstrates the effectiveness of centralised ϵ-greedy begins in the IPD setting and two independent Q-Learning agents described above. In this example, both agents start with the Defect action having a higher value, namely their greedy action. Agents both defecting is the Nash Equilibrium, which makes it a common scenario early on. Afterwards,

Algorithm 4 Centralized ϵ-greedy

```
    Input: List of Agents $\boldsymbol{a}$

1: function SelectActions($\boldsymbol{a}, \epsilon$)
2:     $r \leftarrow Uniform(0,1)$
3:     $\boldsymbol{actions} \leftarrow \emptyset$
4:     for $a_i$ in $\boldsymbol{a}$ do
5:         if $r > \epsilon$ then
6:             $\boldsymbol{actions}.append(a_i.greedy)$
7:         else
8:             $\boldsymbol{actions}.append(a_i.random)$
    return $\boldsymbol{actions}$
```

exploration in an independent ϵ-greedy manner focuses the exploration efforts to the C/D and D/C actions and less often on the more rewarding C/C action. Eq. 4.1 below demonstrates the unwanted asymmetry of the explored joint actions

$$\begin{aligned} P(D,D) &= \overbrace{(1-\epsilon)^2}^{\text{No Expl.}} + \frac{1}{2}\overbrace{\epsilon(1-\epsilon)}^{\text{One Expl.}} + \frac{1}{4}\overbrace{\epsilon^2}^{\text{Both Expl.}} \\ P(C,D) &= P(D,C) = \frac{1}{2}(\frac{1}{2}\epsilon(1-\epsilon) + \frac{2}{4}\epsilon^2) \\ P(C,C) &= \frac{1}{4}\epsilon^2 \end{aligned} \tag{4.1}$$

This assymetry is the result of agents not coordinating their exploration. In order for agents to visit C/C, they *both* have to explore simultaneously, which is rare, especially in smaller values of ϵ. However, with the simple modification of centralised ϵ-greedy, the probabilities of each joint action becomes

$$\begin{aligned} P(D,D) &= \overbrace{(1-\epsilon)}^{\text{No Expl.}} + \frac{1}{4}\overbrace{\epsilon}^{\text{Expl.}} \\ P(C,D) &= P(D,C) = \frac{1}{4}\epsilon \\ P(C,C) &= \frac{1}{4}\epsilon \end{aligned} \tag{4.2}$$

The exploration action, with probability ϵ, uniformly explores the action space.

4.2 Approximate Bayesian Actors for Exploration in MAS

The ϵ-greedy exploration method has been studied widely in the past [10, 47, 52], and produces satisfactory results despite its simplicity. However, ϵ-greedy is built to explore the action space uniformly and with no mechanism that allows preference to unexplored or promising actions. There exist alternatives that address these issues, such as UCB, Boltzmann and Thomson sampling [49]. Thomson sampling (Sec. 3.1.3) has also been applied in deep settings [17] and consists of randomly sampling from posteriors and then maximising the expected reward with respect to the sample.

Traditionally, Thomson sampling has been studied and applied to the multi-armed bandit problem [1, 49], formally an MDP with one state, but can be trivially extended to multiple states [2, 9, 17]. In Q-learning, an agent must maintain posteriors over Q-values, and during action selection, it has to *sample* a Q-value for each action and select the action that maximises the sampled values. This method selects actions with high potential and offers some theoretical guarantees such as self-correcting behaviour and confidence bounds.

Despite this obvious advantage, Thomson sampling is not often used because of the computationally hard task of updating posteriors. Deep learning lacks the framework for Bayesian updates, and Gaussian processes cannot process the amount of data modern RL tasks need. However, recently Gal and Ghahramani [17] showed that the use of dropout in a model is equivalent to Bayesian approximation of the model output, and can be used in conjunction with Thomson sampling for more efficient exploration.

Currently, state of the art in multi-agent RL revolves around Actor-Critic architectures, and adapting them to Thomson sampling is not straightforward. In this section, we extend the state of the art algorithm MADDPG [29] discussed in Sec. 3.3 and make use of dropout as proposed by Gal and

Ghahramani [17] for exploration. Our adaptation will be refered to as MADDPG with Thomson Sampling or MADDPG-TS.

Dropout, during training, randomly disables nodes from the neural network with a probability p. The forward pass and the backpropagation are performed as if the disabled nodes do not exist. If dropout is disabled, each layers output is scaled down by multiplying with p. As mentioned before, in Sec. 2.3, Gal and Ghahramani reinterpret dropout as a Bernoulli distribution and the output of the network as a Bayesian approximation.

MADDPG uses centralized critics, conditioned on all observations and actions $o_1, o_2, \ldots, o_n, a_1, a_2, \ldots, a_n$. In addition, n policies μ_{θ_i} (abbreviated μ_i) are trained by maximizing the expected return $J(\theta_i) = \mathbb{E}[G_i]$. The gradients can be calculated using Eq. 3.8.

It is worth noting that by using dropout in the policy networks, as proposed by Gal and Ghahramani [17], we instead fit a distribution of actions to μ_i. While this leads to a non-deterministic policy, we choose to keep the μ notation since the distribution is internal to the deep network, and can also be disabled. This distribution represents our model uncertainty of what actions maximise the value network output.

Despite Gal and Ghahramani [17] using dropout in an implementation of DQN [31], and fitting distributions in the value network, we will keep point estimates of the Q-values. The reason behind this decision is that training the Actor is dependent on $Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N)$ as seen in Eq. 3.8. However, if the value network can only sample from a distribution, an approximate computation of the expectation requires several forward passes.

Fortunately, a distribution over the policies μ_i is sufficient in the actor-critic setting since it envelops the uncertainty of the value network. Naturally, by sampling from μ_i, we are able to explore efficiently (see Thomson sampling Sec. 3.1.3), making use of the uncertainty over which action might maximise the Q-value.

As a final note, it is worth mentioning the non-trivial task of evaluating policies trained with this method. In a typical ϵ-greedy policy, when evaluating, we set ϵ to zero and essentially use the greedy policy. However, this is not exactly possible with the method described in this section. There are three approaches to address this issue, each with its own disadvantages: i. turn off dropout during evaluation, ii. calculate the mean of multiple samples or, iii. keep using the exploration enabled network.

The first option would be the first instinct of someone using dropout as a noise addition, or a regularizer. However, if considered as fitting a distribution in a sense described by Gal and Ghahramani [17], then turning off dropout is equivalent to sampling $z_{i,k}$ ones from a Bernoulli [17], which is rather unlikely. The second option is much more suited, as it follows the reasoning behind using dropout in the first place: fitting a distribution. Calculating an empirical mean is easy and efficient, requiring only a batched forward pass from the network. Nevertheless, we have opted on never disabling exploration (option iii) and always use Thomson sampling. Empirically, we observed better results this way since the additional stochasticity helps agents escape suboptimal states. This is also utilised in the experiments of well known deep RL algorithms such as the DQN [31].

4.3 Extending to Auto-regressive Joint Exploration

In an effort to combine the two aforementioned methods, centralised ϵ-greedy and MADDPG with Thomson Sampling, we propose a joint exploration method inspired from auto-regressive models seen in unsupervised learning [32]. Our algorithm, MADDPG-AR (MADDPG with **A**uto-**R**egressive exploration), trains an additional $n-1$ policies conditioned on both the observation and the actions of other agents. In addition to the MADDPG-TS's

$\mu_{\theta_i}(o_i)$ policy networks, we learn $n-1$ networks

$$\mu_{\theta_i^c}(o_i, a_1, \ldots, a_{i-1}) \forall i \in \{2, \ldots, n\}$$

that output the corresponding actions a_i. We will refer to them as μ_i^c for brevity.

Those networks will be trained simultaneously, but called in sequence to generate a joint action $\boldsymbol{a}$. Specifically, the action a_1 of the first agent will be generated with the policy $a_1 = \mu_1(o_1)$, then the action generated will be used for $a_2 = \mu_2^c(o_2, a_1)$ to create the second action. This can be repeated to generate actions for all agents and the joint action $\boldsymbol{a} = \{a_1, a_2, \ldots, a_n\}$.

The final architecture of MADDPG-AR consists of critic, decentralised policy, and exploration policy networks. First, n critic networks $Q_i^\mu(\boldsymbol{o}, a_1, \ldots, a_n)$ (seen in Fig. 4.2) are used to approximate the state-action value for the i-th agent. These networks are trained similarly to MADDPG using Eq. 3.7, where μ' is the target policy networks with delayed parameters.

For the decentralised execution, n policy networks $\mu_i(o_i)$ will be trained (Fig. 4.3) and eventually used in decentralised settings, which only receive as an input the agents observation. We will be using gradient ascent to maximise the expectation $J(\theta_i^c) = \mathbb{E}[G_i]$, as also seen in Alg. 5 lines 16–17 and Eq. 3.8.

Finally, the additional $n-1$ networks, conditioned on a sequence of actions (seen in Fig. 4.4 and 4.5) are also trained by maximising again $J(\theta_i) = \mathbb{E}[G_i]$ or:

$$\begin{aligned}\nabla_{\theta_i^c} J\left(\boldsymbol{\mu}_i^c\right) = \mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}} \Big[& \nabla_{\theta_i^c} \boldsymbol{\mu}_i^c \left(a_i | o_i, a_1, \ldots, a_{i-1}\right) \\ & \nabla_{a_i} Q_i^{\boldsymbol{\mu}} \left(\mathbf{x}, a_1, \ldots, a_n\right)|_{a_i = \boldsymbol{\mu}_i^c(o_i, a_1, \ldots, a_{i-1})} \Big]\end{aligned} \tag{4.3}$$

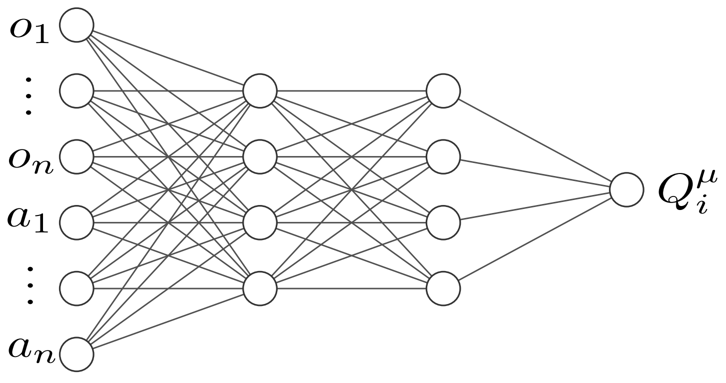


FIGURE 4.2: Critic network. n such networks are used, generating the state-action value for the respective agent. The critic follows the centralised learning paradigm by being conditioned on observations that will not be available in decentralised settings.

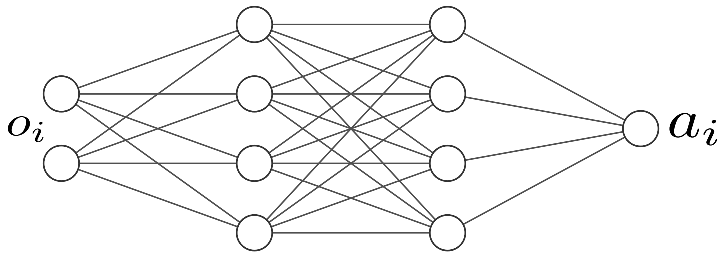


FIGURE 4.3: Actor network. n such networks are used, generating the actions of each agent in a decentralised manner.

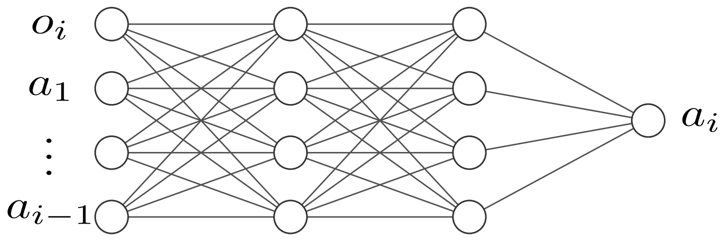


FIGURE 4.4: Coordinated actor network. $n-1$ such networks are used in sequence, generating a joint action used only for exploration during learning.

seen in lines 18–19, Alg. 5. The conditional network will only be used during exploration (lines 4–6) with the purpose of generating experience for the replay buffer D.

Our algorithm is seen in Alg. 5. By replacing the action selection of the original MADDPG (Alg. 3), lines 3–6 generate a joint action in a coordinated fashion. Also, in lines 18–19 we train the additional parameters of the conditional networks.

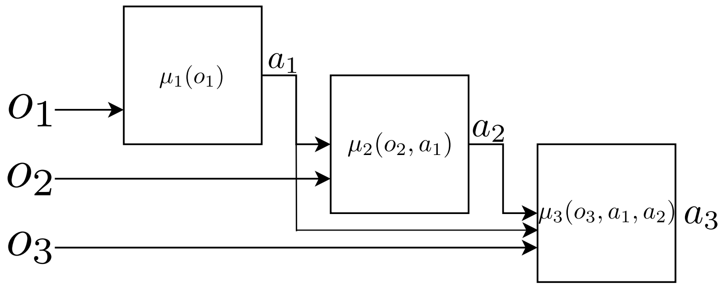


FIGURE 4.5: A top down view of MADDPG-AR action generation with three agents. The first agent only receives its observation, but any subsequent agents also receive the actions of ‘previous’ agents.

Algorithm 5 Coordinated Exploration (in MADDPG) Algorithm

```
    Initialize replay memory $D$ with capacity $N$
    Initialize policies $\theta_{1\ldots n}$ and Q-function parameters $\phi_{1\ldots n}$ for $n$ agents
    Initialize conditional policies $\theta^c_{2\ldots n}$
    Initialize target network with weights $\hat{\theta}_i = \theta_i$ and $\hat{\phi}_i = \phi_i$ for $i = 1 \ldots n$
    Initialize $0 < \tau < 1$ as the target update hyperparameter

 1: for timestep $t$ in every episode $1 \ldots M$ do
 2:     Observe $\boldsymbol{o}$
 3:     $\boldsymbol{a} \leftarrow \{\}$
 4:     Append $a_1 = \mu(o_i; \theta_i))$ to $\boldsymbol{a}$
 5:     for $i$ in $\{2 \ldots n\}$ do
 6:         Append $a_i = \mu(o_i, a_1, \ldots, a_{i-1}; \theta^c_i)$ to $\boldsymbol{a}$    ▷ coordinated actions
 7:     Execute $\boldsymbol{a}$ and observe $\boldsymbol{o'}$, $\boldsymbol{r}$ of all agents
 8:     Store $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{o'}, \boldsymbol{r})$ in buffer
 9:     if $t \mod T_{update} \neq 0$ then
10:         Continue
11:     for $i$ in $\{1 \ldots n\}$ do
12:         Sample mini-batch $B$: $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{o'}, \boldsymbol{r})$ from $D$
13:         $y \leftarrow \begin{cases} r_i, & \text{if } o'_i \text{ is terminal} \\ r_i + \gamma Q(\boldsymbol{o'}, \boldsymbol{a}; \hat{\phi}_i), & \text{otherwise} \end{cases}$
14:         $Q_{gradients} \leftarrow \nabla_{\phi_i} \frac{1}{|B|} \sum_B (Q(\boldsymbol{o}, \boldsymbol{a}; \phi_i) - y)^2$
15:         Gradient descend step on $Q_{gradients}$ with respect to $\phi_i$
16:         $\mu_{gradients} \leftarrow -\frac{1}{|B|} \sum_B \nabla_{a_i} Q(\boldsymbol{o}^B, a^B_1 \ldots \alpha^B_n; \phi_i) \nabla_{\theta_i} \mu(o^B_i; \theta_i)$
17:         Gradient descend step on $\mu_{gradients}$ with respect to $\theta_i$
18:         $\mu^c_{gradients} \leftarrow -\frac{1}{|B|} \sum_B \nabla_{a_i} Q(\boldsymbol{o}^B, a^B_1 \ldots \alpha^B_n; \phi_i) \nabla_{\theta^c_i} \mu(o^B_i, a_1, \ldots, a_{i-1}; \theta^c_i)$

19:         Gradient descend step on $\mu^c_{gradients}$ with respect to $\theta^c_i$
20:         $\hat{\phi}_i \leftarrow \tau \hat{\phi}_i + (1 - \tau) \phi_i$    ▷ Soft updates simi-
21:         $\hat{\theta}_i \leftarrow \tau \hat{\theta}_i + (1 - \tau) \theta_i$    ▷ lar to MADDPG
22:         $\hat{\theta}^c_i \leftarrow \tau \hat{\theta}^c_i + (1 - \tau) \theta^c_i$    ▷ for target stability
```

It should be noted that MADDPG is not mandatory but just a design decision. Our algorithm can be adapted to fit other RL algorithms such as Soft Actor-Critic [22].

4.3.1 Specifying the Order of Action Generation

An immediate question arising from this method is the order in which the actions are generated. However, we argue that the order is not as important and can be arbitrarily chosen. For instance, the PixelCNN [32] model from which this method was inspired, is generating pixels from the upper left to the lower right corner (a randomly chosen order) using this exact procedure and produces exceptional results.

This can also be understood better if one considers that this method attempts to sample from a *joint* policy $P(\boldsymbol{a}|o_1, \ldots, o_n)$ (or abbreviated to $P(\boldsymbol{a})$). In the simplest case of $n = 2$ and using the chain rule we can write it as

$$P(a_1, a_2) = P(a_1)P(a_2|a_1) \tag{4.4}$$

Given that the policies we are training are approximating those distributions and we can sample from them $\mu_1(o_1)$ and $\mu_2(o_2, a_1)$, sampling from the joint

$$\boldsymbol{a} \sim P(a_1, a_2) \tag{4.5}$$

is equivalent to sampling from the prior

$$a_1 \sim P(a_1)$$

and then (due to independency) the conditional

$$a_2 \sim P(a_2|a_1)$$

Of course it is now apparent, that first sampling $a_2 \sim P(a_2)$ and then sampling a conditional $a_1 \sim P(a_1|a_2)$ is equivalent to Eq. 4.5.

Finally, there is an alternative to picking an arbitrarily chosen order, Gibbs sampling [18]. Gibbs sampling allows us to approximate sampling from $P(a_1, a_2)$ by having access to both $P(a_1|a_2)$ and $P(a_2|a_1)$. We have not used this method in our experiments since the increased computing requirements do not offer added performance, but it is worth noting that it exists as an alternative.

Chapter 5

Experiments

In this chapter, we present experiments on our methods and show how they compare to baselines. Centralised ϵ-greedy is first implemented in tabular settings, in the toy problem of Iterated Prisoners Dilemma. There, we show that even simple coordination using centralised ϵ-greedy exploration indeed performs better than simple ϵ-greedy. Then, we experiment in large state-space problems (level-based foraging), where deep learning is a requirement, but also noisy and prone to hyperparameter sensitivity. Again, we show that i. centralised ϵ-greedy exploration outperforms ϵ-greedy in some domains, ii. Thomson Sampling achieves higher returns than previously mentioned methods in non-cooperative level-based foraging, and iii. Auto-regressive exploration learns in less training timesteps and achieves higher returns than all other tested algorithms.

5.1 Centralised ϵ-greedy in Prisoners Dilemma

We borrow the IPD problem, described in Sec. 4.1, and use it as a toy problem due to both its simplicity and difficulty.

TABLE 5.1: Parameters used for the IPD experiments and centralised vs. decentralised ϵ-greedy

Parameter	Value
Episode Length	50
Episode Count	10
Seeds	1000
Learning Rate	0.3
Discount	0.7
Epsilon	0.1, . . . , 0.9

While the Iterated Prisoners Dilemma is represented having a single state[1], we still use Q-learning as a stepping stone to more complex environments. Independent Q-Learning [48] is the most straightforward algorithm that can be complemented with both ϵ-greedy or centralised ϵ-greedy and tested on IPD. We train two IQL agents by repeatedly using the update rule (Eq. 3.4) on a single state.

We allowed each pair to run 10 episodes of 50-step IPD with the parameters found in Tab. 5.1. Then, we noted if the Q-values converged and the policy was the optimal C/C joint action. Convergence was established if the change of Q-values in the last episode was below the threshold of 10^{-2}. This process was repeated for 1000 seeds to generate a percentage. In Fig. 5.1, the percentage of convergence to the C/C actions is shown under the two exploration strategies.

We point out that the x-axis represents different values of ϵ. This visualisation shows that improved performance is achieved in all settings and that centralised ϵ-greedy is not equivalent to *more* exploration (larger ϵ). It can be observed that in this problem, more exploration penalises agent visits to C/D and D/C more often and that improvement is achieved instead with more efficient exploration.

[1]In reality, due to the multi-agent nature of the problem, a state adhering to the Markov property would include a history of all interactions. However, due to the exponential increases of state-space, this is never used in MARL.

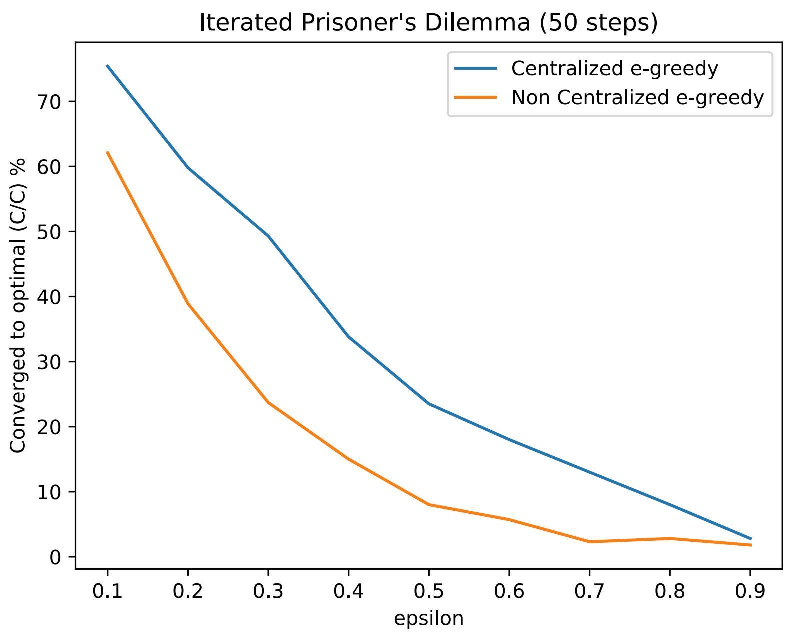
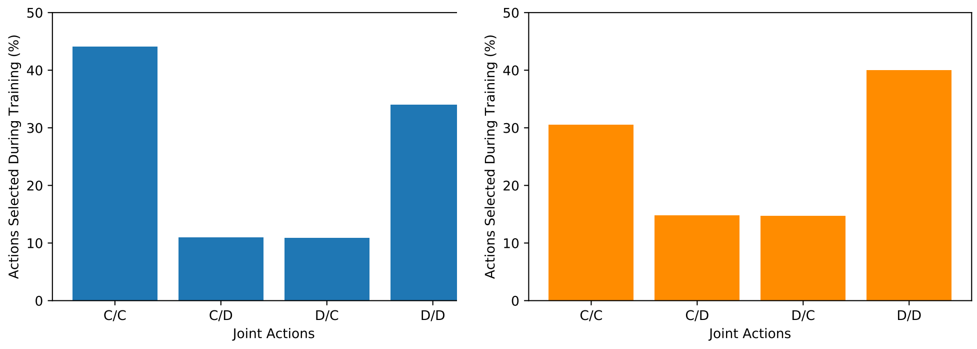


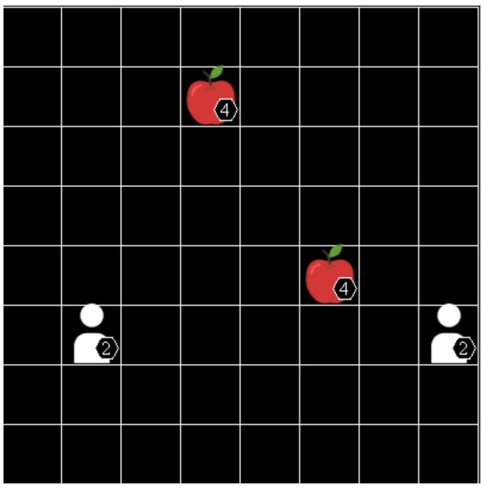
FIGURE 5.1: Performance of two IQL pairs of agents. The agents in the blue pair are using centralised ϵ-greedy while the orange pair are using simple ϵ-greedy. The centralised ϵ-greedy converges more often to the optimal C/C (y axis) regardless of the value of ϵ (x axis).



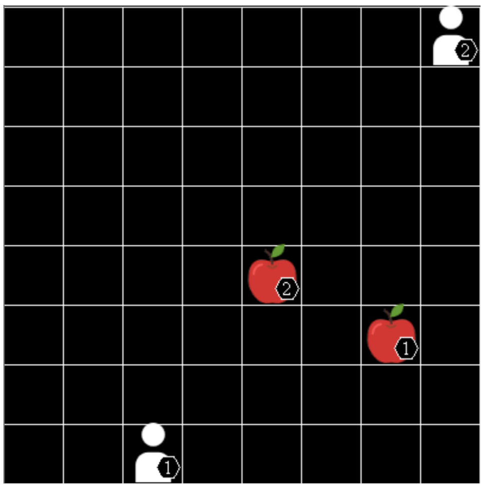
(A) Action selection during learning for centralised ϵ-greedy

(B) Action selection during learning for decentralised ϵ-greedy

FIGURE 5.2: This bar chart shows the distribution of joint actions, during the exploration phase, on the 50 timestep long IPD, with an $\epsilon = 0.2$. As shown, the centralised ϵ-greedy visits the C/D and D/C state less often, which helps converging to the optimal C/C.



(A) An episode where the agents need to cooperate in all instances to collect the food.



(B) The agents should compete, and try to maximise their reward to the detriment of each other.

FIGURE 5.3: A visualisation of the level based foraging domain. The numbers indicate the level of the agent or food. Agents must move adjacent to foods and use the *load* action, while having a cumulative level equal or higher than that of the food. Only agents participating in loading will be rewarded. The levels (a number) are noted next to the foods or agents.

5.2 A Harder Task: Level Based Foraging

IPD proved to be an interesting toy problem that demonstrated that coordinating the exploration efforts is often rewarding. A more interesting and harder environment is Level-Based Foraging, which has been studied in multi-agent systems in the past [3]. This environment is a mixed cooperative-competitive game, which focuses on the coordination of the agents involved. Agents navigate a grid world and collect food by cooperating with other agents if needed.

More specifically, agents are placed in the grid world, and each is assigned a level. Food is also randomly scattered, each having a level on its own. Agents can navigate the environment and can attempt to collect food placed next to them. The collection of food is successful only if the sum of the levels of the agents involved in loading is equal to or higher than the level of the food. Finally, agents are awarded points equal to the level of the food they helped collect, divided by their contribution (their level). Fig. 5.3a and

5.3b show two states of the game, one that requires cooperation, and one more competitive.

While it may appear simple, this is a very challenging environment, requiring the cooperation of multiple agents while being competitive at the same time. In addition, the discount factor also necessitates speed for the maximisation of rewards. Each agent is only awarded points if it participates in the collection of food, and it has to balance between collecting low-levelled food on its own or cooperating in acquiring higher rewards. In situations with three or more agents, highly strategic decisions can be required, involving agents needing to choose with whom to cooperate. Another significant difficulty for RL algorithms is the sparsity of rewards, which causes slower learning.

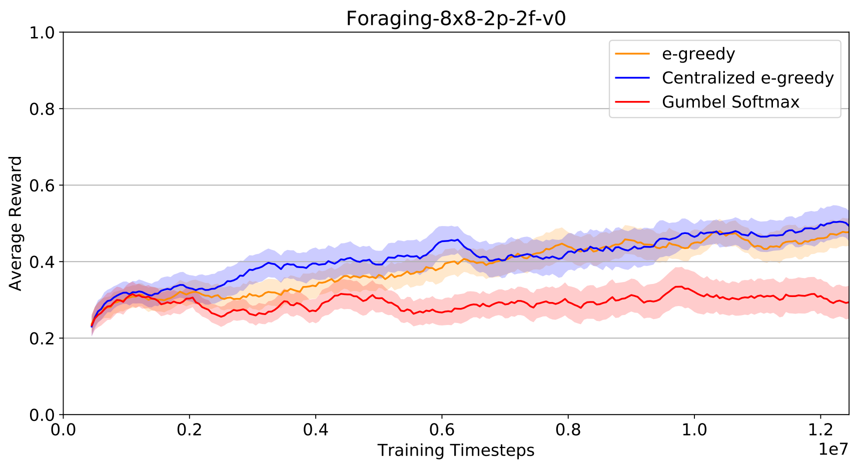
The observation of each agent, is a vector of length $3 * MaxFoodCount + 3 * AgentCount$ where for each food location and agent, the y, x coordinates and the level is listed. The upper-left corner is $(0, 0)$. If a food has been collected then -1 is placed instead of each coordinate. An example of an observation vector, the one of Fig. 5.3a, is printed below:

$$\left[\underbrace{1, 3, 4,}_{y,x,\text{level of food 1}} \quad 4, 5, 4, \quad \underbrace{5, 1, 2,}_{y,x,\text{level of agent 1}} \quad 5, 7, 2 \right]$$

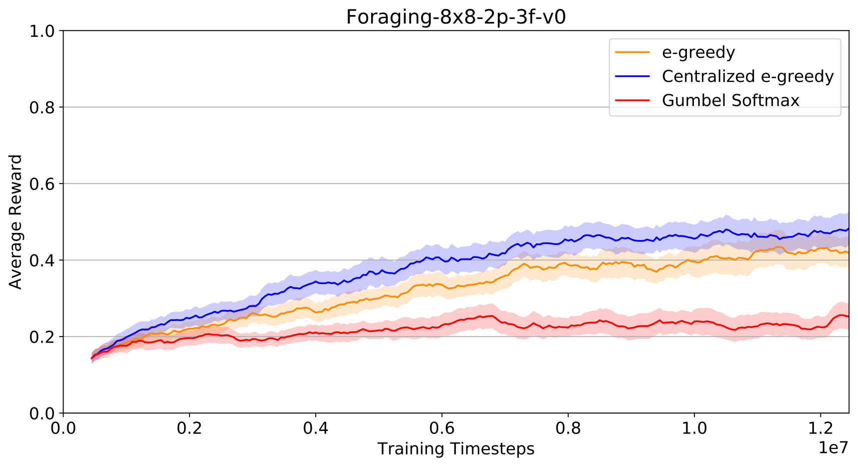
Each agent has access to five actions: $\{North, South, West, East, Load\}$. The first four actions move the agent to that direction, except when another entity is in that title, or it is the edge of the grid, where no movement happens. If the loading action is successful, then the food disappears.

The reward of agent i participating in a successful load action of food f, when the sum of levels of all the starting food is F_T and the set of participating agents is A_T:

$$r_i = \frac{level(i) * level(f)}{F_T * \sum_{j \in A_T} level(j)}$$



(A) Two food locations and two agents in an 8×8 grid variant of level based foraging where foods might or might not require cooperation to load.



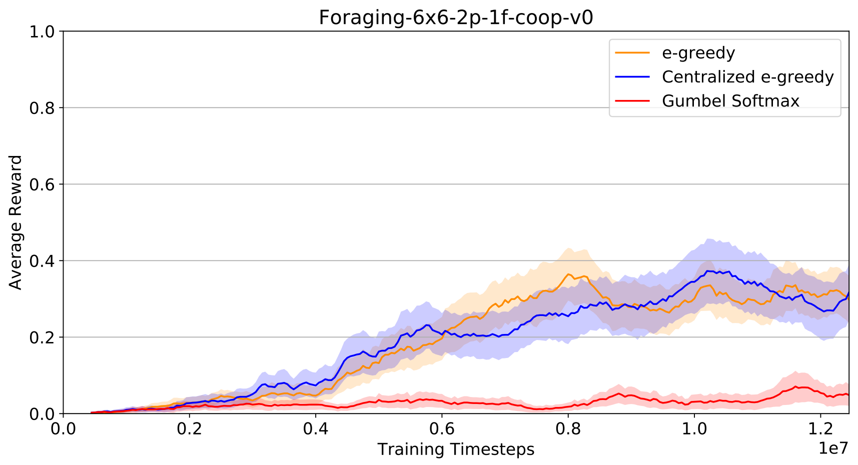
(B) Three food locations and two agents in an 8×8 grid variant of level based foraging where foods might or might not require cooperation to load.

FIGURE 5.4: Performance over training time in the level-based foraging task with two players. Centralised exploration seems to benefit the harder task of three foods.

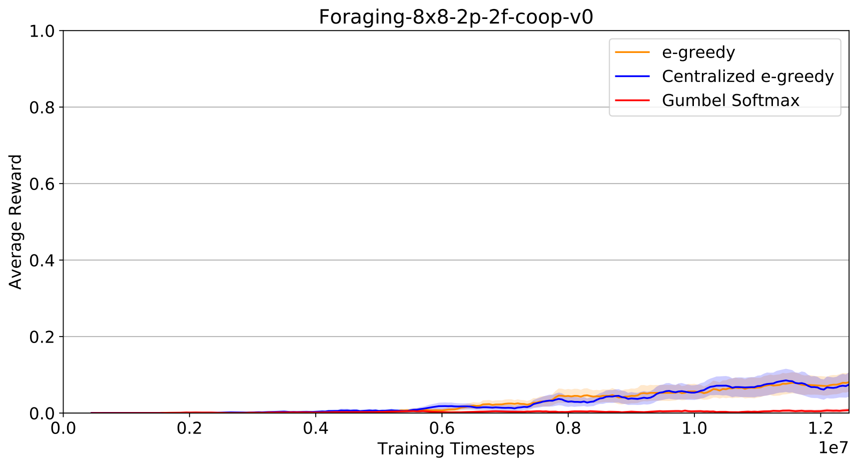
During all other transitions, including failed attempts at collecting food, a reward of zero is given. This reward structure ensures: i. a maximum of one is awarded across all agents in each episode and ii. decisions on what agents to cooperate with, are important.

Our simulator, created as part of this thesis, is a Python implementation for level based foraging[2]. It is based on OpenAI's RL framework, with

[2]https://github.com/semitable/lb-foraging



(A) One food location and two agents in an easier 6×6 grid variant of level based foraging where loading of food *always* requires cooperation



(B) Two food locations and two agents in an 8×8 grid variant of level based foraging where loading of food *always* requires cooperation

FIGURE 5.5: Performance over training time, in the cooperative variants of level-based foraging task with two players. Coordinated foraging turns out to be too hard for sufficient learning to occur. Both variants of ϵ-greedy had similar performance.

modifications for the multi-agent domain. The efficient implementation allows for thousands of simulation steps per second on a single thread, while the rendering capabilities allows humans to visualise agent actions. Our implementation can support different grid sizes or agent/food count. Also, game variants are implemented, such as cooperative mode (agents always need to cooperate) and shared reward (all agents always get the same reward), which is useful as a credit assignment problem.

TABLE 5.2: The evaluation parameters in all experiments of the level-based foraging environment (unless stated otherwise) in the figures presented. During training, 20 evaluation episodes are executed every 5×10^4 timesteps and the average summed reward recorded. The process is repeated with 10 different seeds. Lines in the graph are smoothed with a moving average window of 10.

Eval. Frequency	Episodes	Moving Avg Window	Seeds
5×10^4 timesteps	20	10	10

Given that multi-agent RL has limited availability to highly polished environments, we hope that our implementation will be useful to other RL researchers as well.

In this domain, we will use the term *performance*, *rewards* or *returns* in the context of the sum of the rewards of the agents over an episode. As performance of an algorithm we will be using the average of this sum over all our seeds in an evaluation step (average reward in 20 trial episodes $\times$ 10 seeds unless stated otherwise: Tab. 5.2).

We conducted extensive hyperparameter search on the parameters found in Tab. 5.3 and compared centralised ϵ-greedy on the level based foraging domain. Fig. 5.4 and 5.5 show the performance of ϵ-greedy, centralised ϵ-greedy, and Gumbel Softmax sampling which is the original MADDPG exploration method. Centralised ϵ-greedy has a slight advantage over non-centralised in both Fig. 5.4a (only in the first 6×10^6 timesteps) and 5.4b. However, in the harder cooperative foraging domain (Fig. 5.5), the results appear too noisy, and no apparent improvement is seen. We argue that this is due to the sparse rewards and the necessity of stumbling to a rewarding transition. The figures in this chapter, include in the form of a shaded area the standard error.

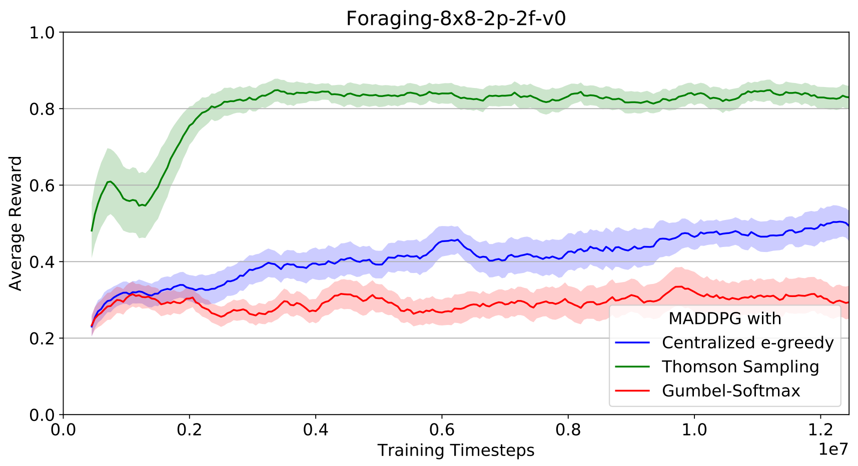
TABLE 5.3: Hyperparameter combinations tested for the two ϵ-greedy algorithms: simple and centralised. In brackets the hyperparameters tested and in bold is the best performing combination of centralised ϵ-greedy (used in the figures below.)

Hyperparameter	Value
Linear Epsilon Anneal (timesteps)	{5e6, **1e7**, 2e7}
Epsilon Target	{**0.01**, 0.1}
Hidden Layers	{[**64,64**], [128,128]}
Actor Learning Rate	{**0.001**, 0.0001}
Critic Learning Rate	{0.01, **0.001**}
Batch Size	1024

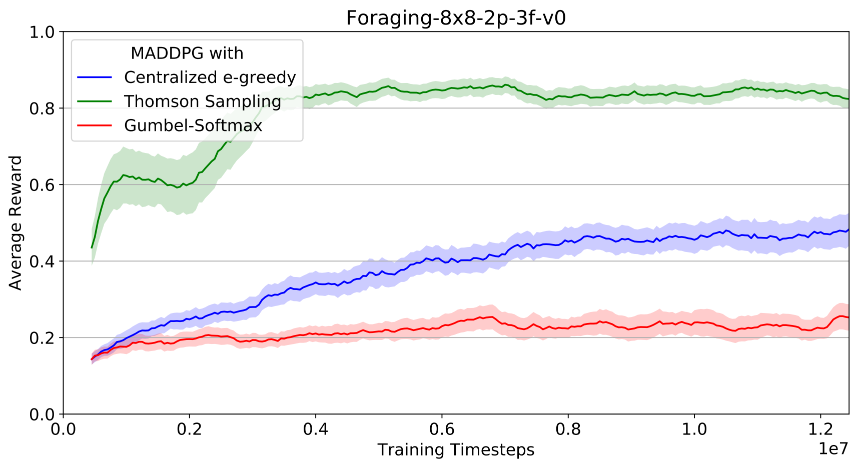
5.3 Improving Performance with Thomson Sampling

We have implemented Thomson sampling exploration as described in Sec. 4.2 by using dropout coupled with MADDPG. As described previously, MADDPG has an Actor-Critic architecture with multiple networks, and which ones should model uncertainty becomes an important decision.

First, the target networks for both the policies and values are supposed to be offering stability during training [29, 31]. We have opted out of using dropout on them, since it inherently creates noisy targets. The next decision is modelling uncertainty on the value networks or the actor networks. While the most natural option would be to be uncertain about the values of the states (the value network), the Actor network is the driving force behind the exploration. We also need to take into account the backpropagation step, where the Actor is trying to maximise the Q-values and is reliant on the critic: a noisy critic would lead to instability in training of the Actor (Eq. 3.8). Finally, since the Actor network is trying to maximise the Q-values, it should have a notion of those values in the hidden layers. Thus, it is reasonable to only enable dropout in the Actor network, and sample from its distribution in order to use Thomson Sampling.



(A) Two food locations and two agents in an 8 × 8 grid variant of level based foraging where foods might or might not require cooperation to load.

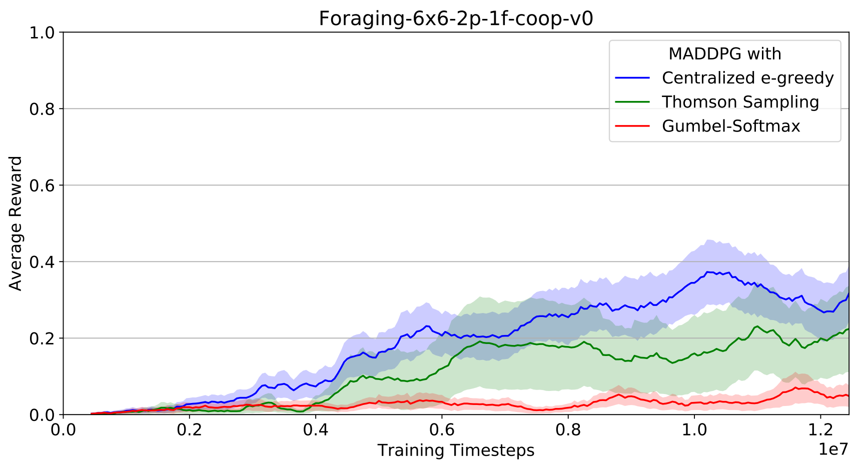


(B) Three food locations and two agents in an 8 × 8 variant of level based foraging where foods might or might not require cooperation to load.

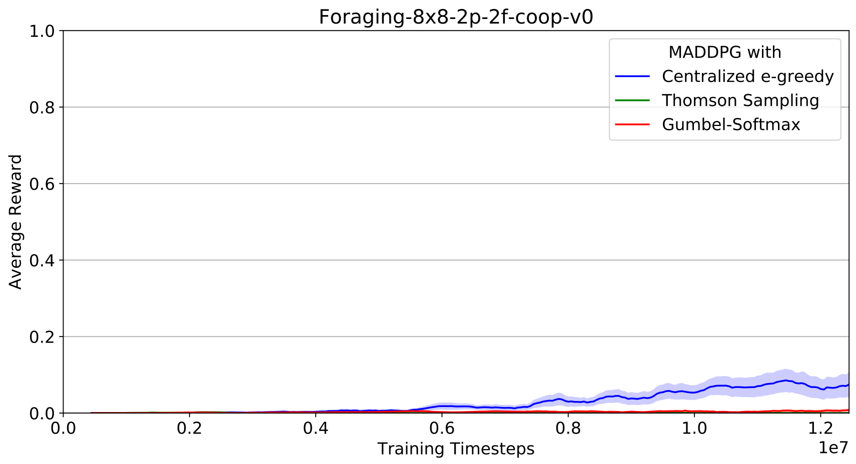
FIGURE 5.6: Performance of Thomson Sampling over training time, in the level-based foraging task with two players. Thomson Sampling exploration greatly outperforms both ϵ-greedy and the original Gumbel Softmax sampling.

Fig. 5.6 show the performance of Thomson sampling with the use of dropout in the level-based foraging domain. Thomson sampling as described in Sec. 4.2 clearly outperforms all other exploration methods and almost reaches an excellent mean reward close to 0.9, which means that 90% of the time all foods are collected. Notably, Thomson sampling only requires 3×10^6 timesteps, a visible improvement over the alternatives.

However, in the harder task of cooperative foraging, where cooperation



(A) One food location and two agents in an easier 6×6 grid variant of level based foraging where loading of food *always* requires cooperation



(B) Two food locations and two agents in an 8×8 grid variant of level based foraging where loading of food *always* requires cooperation

FIGURE 5.7: Performance of Thomson Sampling over training time, in the cooperative variants of level-based foraging task with two players. Thomson Sampling exploration fails to perform as well as centralised ϵ-greedy, which also performs poorly.

is *always* required to load food, this method fails to perform as well as centralised ϵ-greedy (seen in Fig. 5.7). This can be explained if one considers the extreme sparsity of the rewards from the environment. In an $8x8$ grid, the probability of randomly loading a food through cooperation is extremely low. As such, most of the transitions gathered have a reward of zero. Thomson sampling relies on uncertainty, but a plethora of transitions with zero reward lead the networks to have high confidence that all the rewards are indeed none. Because of the narrow distributions, Thomson sampling stops exploring much in contrast to ϵ-greedy where exploration is forced through an ϵ hyperparameter. Thomson sampling could possibly be improved, in these settings, with the use of Prioritised Experience Replay [40].

5.4 Evaluating Auto-regressive Joint Exploration

Finally, we will be completing the experiments with an evaluation of the method proposed in Sec. 4.3. The use of a joint policy dedicated to exploration is novel, and as seen in our results below, outperforms the non-coordinated MADDPG-TS method.

First, we attempt to validate our intuition: that the coordinated network can reliably acquire better rewards. Fig. 5.8, illustrates the rewards of actions, during training, generated through the decentralised policies $\mu(a_1), \mu(a_2)$ and the coordinated $\mu(a_1), \mu(a_2|a_1)$, while the networks are trained through our MADDPG-AR method. We discern that generating actions in a coordinated manner consistently achieves better rewards. Also, the decentralised policies quickly follow and learn from the experience generated.

To compare this method with others, we performed again a grid search of hyperparameters, seen in Tab. 5.4. We used a wider network (that is known to benefit larger dropout values), and a more targeted search for good hyperparameters, which explains the increased performance to Sec. 5.3.

In addition, since this exploration method solves the less demanding variants of level-based foraging quickly, we have replaced the one food variation with four foods on a 10×10 board. For this part of the experiments, we ran a grid search on the hyperparameters of Tab. 5.4 and used the best combination for each of the two algorithms.

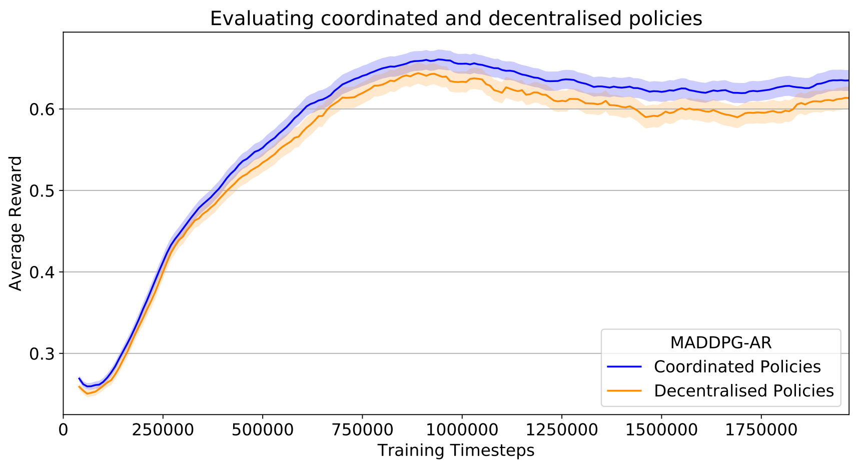


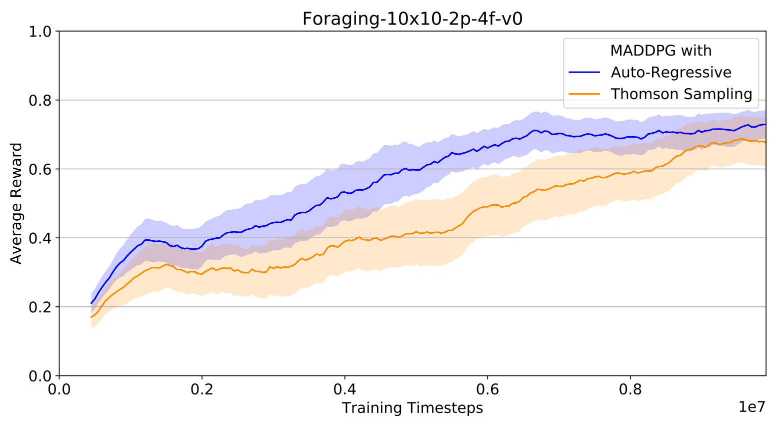
FIGURE 5.8: Evaluation of the decentralised policies: $\mu(a_1), \mu(a_2)$ vs. the coordinated: $\mu(a_1), \mu(a_2|a_1)$. Exploration was performed with coordinated policies (MADDPG-AR). Two player, 8×8 level-based foraging with four food locations.

TABLE 5.4: Hyperparameters searched for the Auto-regressive exploration algorithm and MADDPG-TS. In brackets are the tested hyperparameters and bold highlight shows those that were found to perform the best, and are used in the graphs below.

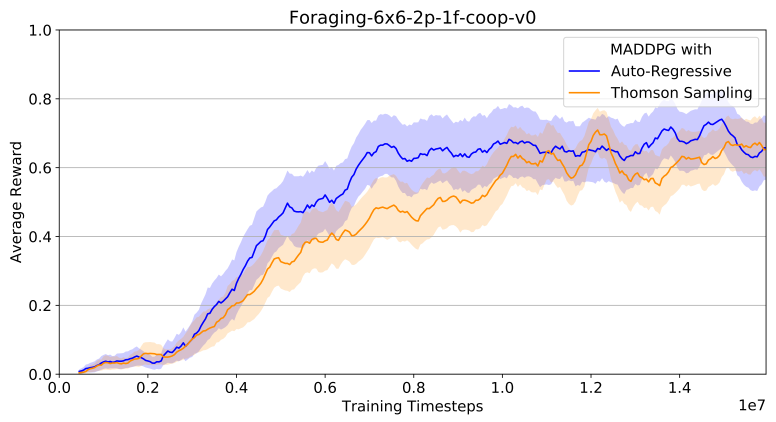
Hyperparameter	Value
Dropout	$\{\mathbf{0.1}, 0.2, 0.5\}$
Hidden Layers	$[100, 100]$
Actor Learning Rate	0.001
Critic Learning Rate	$\{0.01, \mathbf{0.001}\}$
Batch Size	1024

We should note that to generate Fig. 5.9b, we instead used 20 seeds but discarded those that failed to learn anything useful. To ensure fairness, we removed failed seeds from both MADDPG-TS and MADDPG-AR attempts. This was done because this environment needs several random moves to be executed before the agent starts learning and generating meaningful experience, and unfortunately, that was not always the case. The problem is exacerbated because the Actor networks become gradually more confident that all moves lead to zero rewards, thus exploring progressively less.

In Fig. 5.9 we can observe the learning performance using the two distinct



(A) Four food locations and two agents in a larger 10×10 grid



(B) One food location and two agents in an easier 6×6 grid variant of level based foraging where loading of food *always* requires cooperation

FIGURE 5.9: Performance over training time, of MADDPG-TS and MADDPG-AR exploration. The improvement in the two harder tasks is apparent when using coordinated exploration.

exploration methods. We use the exploration proposed previously, MADDPG with Thomson Sampling in Sec. 4.2, which is the adaptation of Gal and Ghahramani [17] in multi-agent systems. MADDPG-AR is our coordinated exploration method proposed in Sec. 4.3. In Fig. 5.9a and 5.9b we can discern that the coordinated exploration method achieves peak performance in significantly less training steps. For clarity, we state again that these are evaluations of the *non-coordinated* policies trained with a coordinated exploration method.

Specifically, in the four food locations environment (Fig. 5.9a) the coordinated method needs less than 7×10^6 timesteps to reach peak performance,

TABLE 5.5: A summary of the maximum performance of all algorithms and environments tested. This is the maximum reward after averaging the seeds and limited to the timesteps shown in the respective graphs. Left column is the environment parameters: grid-size, number of foods and if cooperative. Maximum achievable reward is 1. The standard error can be found in the respective figures in Chapter 5.

	MADDPG	ϵ-greedy	Centr. ϵ-greedy	MADDPG-TS	MADDPG-AR
8x8 2f	0.334	0.480	0.505	**0.848**	-
8x8 3f	0.256	0.434	0.483	**0.861**	-
10x10 4f	-	-	-	0.687	**0.729**
6x6 1f coop	0.071	0.365	0.373	0.354	**0.452**
8x8 2f coop	0.008	0.081	**0.086**	0.003	0.004

TABLE 5.6: Iterations per second for each of our tested algorithms. A network with 64×64 hidden size was used and the level-based foraging domain on a 8×8 grid and two players. Timings were recorded on a current-gen laptop CPU.

	MADDPG	ϵ-greedy	Centr. ϵ-greedy	MADDPG-TS	MADDPG-AR
Speed $\frac{it}{s}$	702.09	1295.77	1289.01	796.11	729.87

while the non-coordinated version needs more than 10^7. That is also similar to the cooperative environment (Fig. 5.9b).

The results are consistent with our expectations. The improved exploration method identifies faster *correct* joint actions, which yield rewards, and performs them, filling the experience replay with meaningful transitions. Eventually, the non-coordinated exploration method also generates sufficient experience and slowly matches the performance.

In Tab. 5.5 we summarise the best performance, after averaging the seeds, of all tested algorithm and environment combinations.

5.5 Computational Requirements

We also measure the training speed of all the algorithms that have been used in this work. Tab. 5.6 presents the recorded training speed. These numbers have been generated on a current-generation laptop CPU, and PyTorch implementation running on a single-core.

MADDPG, with the original Gumbel-Softmax exploration method runs the slowest[3]. On the other hand, the ϵ-greedy methods are the fastest since the exploration cost is limited to generating one random number.

The addition of an extra network when transitioning from MADDPG-TS to MADDPG-AR appears to have minimal effect. Specifically, the MADDPG-TS runs at about 796 iterations per second, while the coordinated method 729: a decrease in speed of 8.42%. However, the need of significantly less timesteps to reach peak performance leads to the non-coordinated exploration requiring (as an example, in the cooperative environment Fig. 5.9b) $\frac{1}{796} * 10^7 \approx 3.5\, hours$ of training time, while coordinated exploration $\frac{1}{729} * 7 * 10^6 \approx 2.6\, hours$: a decrease of 25.71% in wallclock time with our method on the harder environments.

[3]Each sample from Gumbel-Softmax requires two calculations of the log function which is computationally expensive

Chapter 6

Conclusion

This thesis introduces, motivates, and experiments on three novel methods of exploration. First, *centralised ϵ-greedy*, which is a simple, yet impactful alteration to the simple ϵ-greedy. Second, we adapt Thomson sampling with the use of dropout to multi-agent Actor-Critic settings and introduce *MADDPG Thomson Sampling*. Finally, we combine the previous two ideas and create *MADDPG Auto-regressive Exploration*, a method that coordinates exploration while still learning individual policies.

We also implement and formalise level-based foraging, a multi-agent environment, using modern RL frameworks. Changes to the reward function promote sophisticated strategies, while the efficient implementation facilitates the fast generation of trajectories needed for deep RL. We include several ways to tune the difficulty such as changing the number of food, grid size, or a cooperative mode where agents must always cooperate.

We present experiments on all three proposed methods, starting from the toy problem of IPD. Then, we run our methods in several variations of level-based foraging, with various difficulty levels.

The experiments first show that centralised ϵ-greedy has an advantage over simple ϵ-greedy in several cases, such as in the IPD or when having many food locations in level-based foraging. Given the popularity of ϵ-greedy, and the simplicity of our centralised variant, improvement of performance with such a minor modification is noteworthy.

MADDPG with Thomson Sampling clearly outperforms previous exploration methods in non-cooperative level-based foraging environments. In some cases, MADDPG-TS even learns to solve environments that the ϵ-greedy or Gumbel-Softmax exploration algorithms could not learn.

Our Auto-regressive Joint Exploration algorithm improves MADDPG-TS even further and reduces the training steps required to reach peak performance. This reduction leads to less computing time requirements (up to a decrease of 25% in training time) over the second-best exploration algorithm, MADDPG-TS.

All these exploration methods were paired with MADDPG, a state of the art multi-agent deep RL algorithm. However, we show that by changing the original exploration method, Gumbel-Softmax sampling, we regularly outperform it significantly in the tested environments.

In the future, we plan on testing MADDPG-AR on more complicated domains such as a robotic warehouse simulator. A large number of agents should benefit even more from a coordinated approach to exploration. Also, we would like to experiment on a coordinated exploration method along with the Soft Actor-Critic (instead of MADDPG), an algorithm that focuses explicitly on exploration.

Bibliography

[1] Shipra Agrawal and Navin Goyal. “Analysis of Thompson Sampling for the Multi-armed Bandit Problem”. In: *COLT 2012 - The 25th Annual Conference on Learning Theory*. Ed. by Shie Mannor, Nathan Srebro, and Robert C. Williamson. Vol. 23. JMLR Proceedings. JMLR.org, 2012, pp. 391–3926.

[2] Shipra Agrawal and Navin Goyal. “Thompson Sampling for Contextual Bandits with Linear Payoffs”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org, 2013, pp. 127–135.

[3] Stefano V. Albrecht and Subramanian Ramamoorthy. “A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems”. In: *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2013, pp. 1155–1156.

[4] Stefano V. Albrecht and Peter Stone. “Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems”. In: *Artificial Intelligence* 258 (2018), pp. 66–95. DOI: 10.1016/j.artint.2018.01.002.

[5] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017 (NeurIPS)*. Ed. by Isabelle Guyon et al. 2017, pp. 5048–5058.

[6] Robert Axelrod. “The Evolution of Strategies in the Iterated Prisoner’s Dilemma”. In: *The Dynamics of Norms* (1987), pp. 1–16.

[7] George W. Brown. “Iterative Solution of Games by Fictitious Play”. In: *Activity Analysis of Production and Allocation* 13.1 (1951), pp. 374–376.

[8] Georgios Chalkiadakis and Craig Boutilier. “Coordination in Multiagent Reinforcement Learning: A Bayesian Approach”. In: *Proceedings of the 2nd International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. ACM, 2003, pp. 709–716. DOI: 10.1145/860575.860689.

[9] Olivier Chapelle and Lihong Li. “An Empirical Evaluation of Thompson Sampling”. In: *Advances in Neural Information Processing Systems 24: Annual Conference on Neural Information Processing Systems 2011 (NeurIPS)*. Ed. by John Shawe-Taylor et al. 2011, pp. 2249–2257.

[10] Richard Dearden, Nir Friedman, and Stuart J. Russell. “Bayesian Q-Learning”. In: *Proceedings of the 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*. Ed. by Jack Mostow and Chuck Rich. AAAI Press / The MIT Press, 1998, pp. 761–768.

[11] Michael O’Gordon Duff and Andrew Barto. “Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes”. PhD thesis. University of Massachusetts at Amherst, 2002.

[12] Yaakov Engel, Shie Mannor, and Ron Meir. “Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML)*. Ed. by Tom Fawcett and Nina Mishra. AAAI Press, 2003, pp. 154–161.

[13] Yaakov Engel, Shie Mannor, and Ron Meir. “Reinforcement Learning with Gaussian Processes”. In: *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. Ed. by Luc De Raedt and

Stefan Wrobel. Vol. 119. ACM International Conference Proceeding Series. ACM, 2005, pp. 201–208. DOI: 10.1145/1102351.1102377.

[14] Jakob N. Foerster et al. "Counterfactual Multi-Agent Policy Gradients". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, the 30th innovative Applications of Artificial Intelligence, and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (AAAI)*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 2974–2982.

[15] Jakob N. Foerster et al. "Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1146–1155.

[16] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1582–1591.

[17] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 1050–1059.

[18] Stuart Geman and Donald Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 6.6 (1984), pp. 721–741. DOI: 10.1109/TPAMI.1984.4767596.

[19] Mohammad Ghavamzadeh and Yaakov Engel. “Bayesian Actor-Critic Algorithms”. In: *Proceedings of the 24th International Conference on Machine Learning (ICML)*. Ed. by Zoubin Ghahramani. Vol. 227. ACM International Conference Proceeding Series. ACM, 2007, pp. 297–304. DOI: 10.1145/1273496.1273534.

[20] Mohammad Ghavamzadeh and Yaakov Engel. “Bayesian Policy Gradient Algorithms”. In: *Advances in Neural Information Processing Systems 19 (NIPS)*. Ed. by Bernhard Schölkopf, John C. Platt, and Thomas Hofmann. MIT Press, 2006, pp. 457–464.

[21] Mohammad Ghavamzadeh et al. “Bayesian Reinforcement Learning: A Survey”. In: *Foundations and Trends in Machine Learning* 8.5-6 (2015), pp. 359–483. DOI: 10.1561/2200000049. arXiv: 1609.04436 [cs.AI].

[22] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1856–1865.

[23] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, 2016, pp. 2094–2100.

[24] Hado van Hasselt et al. “Deep Reinforcement Learning and the Deadly Triad”. In: *CoRR* abs/1812.02648 (2018). arXiv: 1812.02648.

[25] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and Acting in Partially Observable Stochastic Domains”. In: *Artificial Intelligence* 101.1-2 (1998), pp. 99–134. DOI: 10.1016/S0004-3702(98)00023-X.

[26] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. “Bayesian Reinforcement Learning in Factored POMDPs”. In: *CoRR* abs/1811.05612 (2018). arXiv: 1811.05612.

[27] Jesse Levinson et al. “Towards Fully Autonomous Driving: Systems and Algorithms”. In: *IEEE Intelligent Vehicles Symposium (IV), 2011, Baden-Baden, Germany, June 5-9, 2011*. IEEE, 2011, pp. 163–168. DOI: 10.1109/IVS.2011.5940562.

[28] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[29] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *Advances in Neural Information Processing Systems 30 (NIPS)*. Ed. by Isabelle Guyon et al. 2017, pp. 6382–6393.

[30] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016, pp. 1928–1937.

[31] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602.

[32] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel Recurrent Neural Networks”. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 1747–1756.

[33] Georgios Papoudakis et al. “Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning”. In: *CoRR* abs/1906.04737 (2019). arXiv: 1906.04737.

[34] Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. “An On-line POMDP Algorithm for Complex Multiagent Environments”. In: *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Ed. by Frank Dignum et al. ACM, 2005, pp. 970–977. DOI: 10.1145/1082473.1082620.

[35] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. “Point-based Value Iteration: An Anytime Algorithm for POMDPs”. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Georg Gottlob and Toby Walsh. Morgan Kaufmann, 2003, pp. 1025–1032.

[36] Pascal Poupart et al. “An Analytic Solution to Discrete Bayesian Reinforcement Learning”. In: *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. Ed. by William W. Cohen and Andrew Moore. Vol. 148. ACM International Conference Proceeding Series. ACM, 2006, pp. 697–704. DOI: 10.1145/1143844.1143932.

[37] Tabish Rashid et al. “QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4292–4301.

[38] Stéphane Ross, Brahim Chaib-draa, and Joelle Pineau. “Bayes-Adaptive POMDPs”. In: *Advances in Neural Information Processing Systems 20 (NIPS)*. Ed. by John C. Platt et al. Curran Associates, Inc., 2007, pp. 1225–1232.

[39] Stéphane Ross and Joelle Pineau. “Model-Based Bayesian Reinforcement Learning in Large Structured Domains”. In: *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI)*. Ed. by David A. McAllester and Petri Myllymäki. AUAI Press, 2008, pp. 476–483. arXiv: 1206.3281 [cs.AI].

[40] Tom Schaul et al. “Prioritized Experience Replay”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[41] Sandip Sen, Mahendra Sekaran, and John Hale. “Learning to Coordinate without Sharing Information”. In: *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*. Ed. by Barbara Hayes-Roth and Richard E. Korf. AAAI Press / The MIT Press, 1994, pp. 426–431.

[42] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. “Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving”. In: *CoRR* abs/1610.03295 (2016). arXiv: 1610.03295.

[43] Yoav Shoham, Rob Powers, and Trond Grenager. “If Multi-Agent Learning Is the Answer, What Is the Question?” In: *Artificial Intelligence* 171.7 (2007), pp. 365–377. DOI: 10.1016/j.artint.2006.02.006.

[44] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815.

[45] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529 (2016), pp. 484–503.

[46] Trey Smith and Reid G. Simmons. “Heuristic Search Value Iteration for POMDPs”. In: *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence (UAI)*. Ed. by David Maxwell Chickering and Joseph Y. Halpern. AUAI Press, 2004, pp. 520–527.

[47] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning - An Introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 0262193981. DOI: 10.1109/tnn.1998.712192.

[48] Ming Tan. “Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents”. In: *Proceedings of the 10th International Conference on Machine Learning (ICML)*. 1993, pp. 330–337.

[49] William R Thompson. “On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples”. In: *Biometrika* 25.3/4 (1933), pp. 285–294.

[50] Oriol Vinyals et al. “StarCraft II: A New Challenge for Reinforcement Learning”. In: *CoRR* abs/1708.04782 (2017). arXiv: 1708.04782.

[51] Christopher John Cornish Hellaby Watkins. “Learning from Delayed Rewards”. PhD thesis. King’s College, Cambridge, 1989.

[52] Michael Wunder, Michael L. Littman, and Monica Babes. “Classes of Multiagent Q-learning Dynamics with epsilon-greedy Exploration”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 1167–1174.

[53] Peter R. Wurman, Raffaello D’Andrea, and Mick Mountz. “Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses”. In: *AI Magazine* 29.1 (2008), pp. 9–20.