Multi-Agent Deep Reinforcement Learning: Revisiting MADDPG

Joshua John Wilkins

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2021

Abstract

Multi-agent reinforcement learning (MARL) is a rapidly expanding field at the forefront of current research into artificial intelligence. We examine MADDPG, one of the first MARL algorithms to use deep reinforcement learning, on discrete action environments to determine whether its application of a Gumble-Softmax impacts its performance in terms of average and maximum returns. Our findings suggest that while Gumbel-Softmax negatively impacts performance, the deterministic policy that necessitates its use performs far better with the multi-agent actor-critic method than the stochastic alternative we propose without Gumbel-Softmax.

Acknowledgements

I want to thank everyone at the Autonomous Agents Research Group for allowing me to join you during my master's thesis. I would also like to thank Filippos for his supervision throughout the summer.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Joshua John Wilkins*)

Table of Contents

Chapter 1

Introduction

Multi-agent reinforcement learning (MARL) defines a method whereby multiple agents repeatedly interact with the same environment to solve a given multi-agent task (e.g. [10]). Proposed by Lowe et al. [26], MADDPG is one of the first MARL algorithms to use deep reinforcement learning and, since its introduction, is widely used as a baseline in MARL research (e.g. [9]).

MADDPG combines the multi-agent actor-critic (MAAC) method with the DDPG algorithm [24]. Lowe et al. [26] introduced the MAAC method to address the challenges faced by fully decentralised algorithms such as IQL and IAC within multi-agent tasks. Their MAAC implementation differs from these algorithms because it allows information to be shared between agents during training, creating a stationary environment for the individual agents [26].

Lowe et al. [26] found that MADDPG could learn cooperative behaviours in multi-agent tasks where its decentralised counterparts could not. Thus, their work suggests that the MAAC method has advantages over fully decentralised methods. Unfortunately, Lowe et al. [26] did not explain their decision to implement MAAC with DDPG.

The DDPG algorithm is designed for continuous actions. Therefore, Lowe et al. [26] employ a Gumbel-Softmax to ensure that MADDPG would work for discrete actions [21]. However, recent work has hypothesised that Gumbel-Softmax introduces reparameterisation bias, worsening MADDPG's performance in terms of average returns [35].

Our work explores the researchers' hypothesis that Gumbel-Softmax hinders MADDPG's performance in discrete-action spaces and is motivated by the supposition that an alternative implementation of the MAAC method with a discrete-action RL algorithm would outperform MADDPG on multi-agent discrete-action tasks in terms of

average returns or at least perform similarly while being easier to tune.

Stated clearly, we investigate the following research questions: (1) Does Gumbel-Softmax hinder MADDPG's performance for discrete actions? (2) Can we create an alternative MAAC algorithm, designed for discrete actions, that outperforms MADDPG in terms of returns or ease of use?

Therefore, we propose MAPPOQ, an algorithm that implements the MAAC method with the PPO RL algorithm. We compared the performances of MAPPOQ, MADDPG, and several baselines across a diverse range of multi-agent tasks. Our results suggest that there exists a more nuanced trade-off than we had first expected between implementing MAAC with DDPG and a Gumbel-Softmax vs implementing MAAC with an RL algorithm designed for discrete action spaces.

While Gumbel-Softmax did appear to cause worse performance for MADDPG than MAPPOQ on the less complex discrete tasks, as the complexity increased, MADDPG outperformed MAPPOQ. Thus, suggesting that the reduction in policy variance from DDPG may outweigh the reparameterisation bias caused by Gumbel-Softmax. These points shall be explored in greater detail in the coming chapters.

The remainder of the thesis is structured into six separate chapters. First, the Background chapter, where we cover the necessary pre-requisites for understanding the project. Next, the Related Work chapter outlines several similar approaches in the literature, and we explain how our algorithm is different. After, the Methods and Results chapters provide details of the experiments we carried out. Then within the Discussion chapter, we expand on the results and the project as a whole, outlining limitations and our speculations. Finally, in the Conclusion chapter, we state our concluding remarks and provide suggestions for future work.

Chapter 2

Background

The background chapter outlines foundational material required for understanding the project. For an in-depth view on the topics discussed, Richard Sutton and Andrew Barto's *Reinforcement Learning: An introduction* is highly recommended.

2.1 Markov Decision Processes and Value Methods

As will be shown, Markov Decision Processes (MDPs) provide the necessary framework to discuss learning via interaction for a pre-defined goal in the single-agent case. Later in this chapter, we will introduce another framework specifically for the multi-agent case. However, first, it is necessary to understand MDPs, as many of the multi-agent algorithms defined later build upon the details outlined here.

Within an MDP, an agent interacts with an environment at discrete time intervals t. The environment then provides transition information for each of the agent's actions A_t, including the new state of the environment S_{t+1}, and the reward achieved by the agent, R_{t+1}. Consequently, repeated interactions between agent and environment lead to a transition history that appears as $S_0, A_0, R_1, \ldots, S_t, A_t, R_{t+1}$. For convenience, we write the sets of states, actions and rewards within an MDP as $\mathcal{S}, \mathcal{A}, \mathcal{R}$, respectively.

Crucially, MDPs satisfy the Markov property (e.g. [28]), meaning that all information on past agent-environment interactions are contained within the current state [44, p. 49]. Therefore, given the Markov property, MDPs allow us to define a discrete probability distribution by treating the rewards and next states as random variables, conditioned solely on the previous state and action. The distribution can be written as

follows:

$$p(s',r \mid s,a) \coloneqq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}, \tag{2.1}$$

for all $s',s \in \mathcal{S}, r \in \mathcal{R},$ and $a \in \mathcal{A}(s)$.

Following this, it is possible to calculate the expected value for any given state or state-action pair in the environment [44, p. 58-59]. Moreover, by utilising the Bellman Equations [5], one can define functions that map any state or state-action pair to their respective expected return value for a given probability distribution over the set of possible actions. The probability distribution is referred to as the agent's policy π [44, p. 59]. The state and state-action value functions for a given policy π are defined as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s], \forall s \in \mathcal{S} \tag{2.2}$$

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \tag{2.3}$$

where G_t is the total returns attained after a given timestep t. To handle continuous tasks and to prevent over valuing future returns, a discount factor γ is applied to the rewards. We therefore define the total returns as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.

Value-based methods are a family of methods that utilise the value functions above to solve reinforcement learning (RL) tasks [44]. That is, they attempt to converge on a policy that maximises the value function for all states of a given environment. Such a policy is called the optimal policy π^*. The following section discusses an alternative family of methods where agents learn by improving policies directly without needing to iterate over expected return values.

2.2 Policy Gradient Methods

This section outlines a set of methods that enable direct policy updates.

Policy gradient methods do not use tables to store policy information. Instead, they use a weighted function, called the parameterised policy, that maps states to the agent's associated action probabilities, [45]. The following shows an example for defining a parameterised policy with the softmax function, which is linear in the feature vectors [44, p. 322]:

$$\pi(a \mid s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,a,\theta)}}, \tag{2.4}$$

where $\theta \in \mathbb{R}^{d'}$, $h(s,a,\theta) = \theta^T \mathbf{x}(s,a)$, and $\mathbf{x}(s,a) \in \mathbb{R}^{d'}$ are the feature vectors of the environment.

Policy parameterisation provides the first important step for defining policy gradient methods; the second is provided by the result from the *policy gradient theorem* [45], which states:

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \mathrm{d}a\mathrm{d}s \tag{2.5}$$

$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta} [\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)] \tag{2.6}$$

where $\nabla J(\theta) \in \mathbb{R}^{d'}$ is the gradient of an arbitrary scalar performance measure with respect to θ, and $\rho^\pi(s') := \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) ds$ is the discounted state distribution.

The *policy gradient theorem* proves that to increase the performance of our policy π, as measured by an objective function $J(\theta)$; we can move the parameter θ in a direction proportional to the expected reward and inversely proportional to the probability of taking the action under the policy. Intuitively, this makes sense, as the higher the reward, the more we should choose the same action in the future. Hence, applying gradient ascent to the terms within the expectation will move the policy parameter theta in a direction that maximises the scalar performance measure function—in turn, converging towards the optimal policy [45], [44, pp. 324-327].

Whilst policy gradient methods are a great tool [45], vanilla implementations like the REINFORCE algorithm suffer from high variance in their gradient estimates, and thus slow convergence rates [49]. The issue is primarily caused because of the requirement to use the complete episode returns in the actor's loss function since the repeated sampling of actions from the actor's policy probability distribution adds noise into the gradient estimates [39]. Moreover, the states and rewards are dependent on these action samples. Hence, we effectively accumulate the noise introduced by every action sampled in the episode by using complete episodes returns. Additionally, it is known that the variance in the gradient estimation is linearly proportional to the transition history [33].

The following section discusses a technique that addresses the issues with policy gradient methods, and leads to a new set of methods that are being applied in multiple, more modern RL algorithms (e.g. [31, 40, 24, 35**?**]).

2.3 Actor-Critic Methods

Actor-Critic methods build upon a combination of ideas from both value-based and policy-based learning. Critically, they address the high variance issues that persist in policy gradient methods [23], [44, pp. 331-332]. This section focuses on detailing actor-critic methods and examining why they are used for many RL tasks.

As the name suggests, actor-critic methods come in two parts. Firstly, the actor that maintains a policy and acts in the environment, and secondly, the critic that learns the expected return values with a value-based method. Accordingly, actor-critic methods approximate the total returns of an episode, with the critic updating a temporal difference (TD) target [44, pp. 331-332]. Hence, they replace the complete returns needed for the policy update with the critic's TD target enabling online policy updates [42]. That is, the policy updates can take place before the completion of an entire episode. Below we give the TD target, the critic update and the actor update:

$$\delta_{t+1} = R + \gamma \hat{v}(S', w_t) - \hat{v}(S, w_t), \tag{2.7}$$

$$w_{t+1} = w_t + \alpha^{w_t} \delta_{t+1} \nabla \hat{v}(S, w_t), \tag{2.8}$$

$$\theta_{t+1} = \theta_t + \alpha^{\theta_t} \delta_{t+1} \nabla_{\theta_t} \log \pi_{\theta_t}(a \mid s), \tag{2.9}$$

where, $\hat{v}(S, w_t)$ is the approximate value function parameterised by w, and $\alpha^w, \alpha^\theta > 0$ are the step sizes.

Estimating the complete episode returns with a TD target reduces the number of observed transitions required to update the policy's gradient. Consequently, the TD target reduces the variance caused by sampling actions from the actor's policy distribution for the same reasons discussed at the end of the previous section [44, p. 124]. Moreover, the use of a TD target enables the direct learning of the policy to be extended to continuous tasks where it is not possible to calculate the complete returns because they have no ending. Therefore, actor-critic methods are essential for any algorithm applying policy gradient updates to continuous tasks or tasks with very long episodes [42], [44, p. 124]. Additionally, any algorithm performing policy gradient updates in challenging environments can also benefit from the reduced variance of actor-critic methods [23, 45].

An on-policy RL algorithm updates the current policy with samples it collected. Conversely, off-policy algorithms update with samples collected by a different policy

than the current one [44, pp. 100]. Below are two examples of on-policy actor-critic algorithms.

Synchronous Advantage Actor-Critic (A2C): The on-policy actor-critic algorithm, A2C, utilises the advantage within its policy updates [31]. Specifically, the advantage function is defined as the difference between the expected state-action value and the expected state value. By utilising the Bellman equations, the state-action value function can be rewritten in terms of the state value function, $Q(s_t, a_t) = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$. Therefore, we arrive at the following equation:

$$A_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) - V(s_t), \tag{2.10}$$

where T is some finite time horizon less than the episode length. Hence, by utilising the advantage in place of the complete returns, we can write the gradient of the objective function as:

$$\nabla J_\theta = \mathbb{E}_t \left[\nabla_\theta \pi_\theta(a_t \mid s_t) A_t \right] \tag{2.11}$$

Finally, the A2C algorithm trains synchronously on several parallel environments to ensure independent and identically distributed (i.i.d) transition samples [31].

Proximal Policy Optimisation (PPO): PPO is an alteration of the A2C algorithm [38]. It replaces the log action probabilities in the A2C policy update with an importance sampling ratio between the old and new policies. Additionally, Schulman et al. [38] clip the importance ratio to prevent the updates to the policy gradient from being too large. Hence, Schulman et al. [38] propose the following PPO objective function:

$$\nabla J_\theta = \mathbb{E}_t \left[\min(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) A_t) \right], \tag{2.12}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, and A_t is the general advantage estimation (GAE), given by:

$$A_t = \sigma_t + (\gamma\lambda)\sigma_{t+1} + (\gamma\lambda)^2 \sigma_{t+2} + \cdots + (\gamma\lambda)^{T-t+1} \sigma_{T-1}, \tag{2.13}$$

for $\sigma_t = r_t + \gamma V(s_{t+1}) - V(s_t)$.

Lastly, the PPO algorithm performs multiple training updates with the same advantage estimation. In this way, PPO achieves higher sample efficiency than other on-policy algorithms [38, 52].

The following section examines a policy gradient method where action selection is no longer stochastic.

2.4 Deterministic Policy Gradient

Introduced by Silver et al. [40], Deterministic Policy Gradient (DPG) is a variant of policy gradient whereby actions are no longer sampled from a stochastic policy conditioned on the state, e.g. $a = \pi(.|s)$, and instead are selected deterministically, $a = \mu(s)$. This section explores how the DPG method works, and why according to the authors, it significantly improves upon stochastic policy gradient (SPG) methods in environments with high dimensional action spaces.

Although we no longer sample actions in DPG from a stochastic policy, it is still possible to calculate the policy gradient [40]. Accordingly, we integrate solely over the state space distribution since the actions are deterministic. This is relevant to the performance claims for DPG made by Silver et al. [40], which we will discuss below. Formally, the deterministic objective function is as follows:

$$J(\mu_\theta) = \int_{\mathcal{S}} \rho^{\mu}(s) r(s, \mu_\theta(s)) \mathrm{d}s \tag{2.14}$$

$$= \mathbb{E}_{s\sim\rho^{\mu}}\left[r(s, \mu_\theta(s)\right]. \tag{2.15}$$

Subsequently, the deterministic policy gradient theorem then provides the following result:

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu}(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s \tag{2.16}$$

$$= \mathbb{E}_{s\sim\rho^{\pi}, a\sim\pi_\theta}[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu}(s,a)|_{a=\mu_\theta(s)}] \tag{2.17}$$

As shown in (2.16), DPG methods do not integrate over the action space. This is in contrast to SPG methods, which require two integrals to calculate the objective function–one over the action space and one over the observation space as shown in equation (2.5). In their paper, Silver et al. [40] show that as a result of this, DPG has an improved computational cost over SPG as the number of actions in the action space grows. Moreover, Silver et al. [40] also show that DPG significantly outperforms its stochastic counterpart in high-dimensional action spaces, since SPG requires many more environmental samples to estimate the policy gradient. A possible drawback for DPG is that it loses the inherent exploration from sampling a policy distribution. However, Silver et al. [40] propose a method to surmount this issue that sees DPG applied off-policy to allow for the use of a stochastic behavioural policy and subsequently increases exploration.

Silver et al. [40] introduce a stochastic behavioural policy that collects transitions from the environment. These transitions are then passed to a deterministic target policy

that updates the policy gradient estimate. Furthermore, Silver et al. [40] highlight that because DPG removes the integral over the actions, importance sampling is not required in the actor update. Additionally, by using Q-learning for the critic, they can also avoid using importance sampling in its update. Specifically, we write the critic update as follows:

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t), \tag{2.18}$$

where

$$\delta_t = r_t + \gamma Q^w(s_t, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t). \tag{2.19}$$

Subsequently, we can derive the following off-policy deterministic policy training update, where compatible function approximation enables us to replace the gradient of the Q^μ with the critic gradient [40]:

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)} \tag{2.20}$$

The following section will focus on an algorithm that builds upon DPG by utilising some of the techniques applied in the Deep Q-learning Network (DQN) algorithm introduced by researchers at DeepMind [30].

2.5 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is an algorithm proposed by Lillicrap et al. [24]. The authors' primary motivation for developing DDPG was to actualise the advantages afforded by DPG methods. They discerned that by combining deterministic policy updates with parts of DQN [30] they could create a novel algorithm that would successfully learn policies in high-dimensional and continuous action spaces [24]. This section will briefly outline their proposed DDPG algorithm.

As mentioned, DDPG builds upon concepts from the DQN algorithm [24, 30]; specifically, these are: (1) using a neural network to approximate state-action values, (2) employing an experience replay buffer to hold training samples, and (3) the use of frozen target networks.

Firstly, neural networks can be trained to approximate the state-action value function. Within high dimensional environments, storing a table of state-action values becomes impractical. Moreover, in the case of continuous action spaces, it is impossible to store such a table [24],[44, pp. 223-228]. Hence, neural networks allow DDPG

to operate as intended—in environments with continuous action spaces. Secondly, an experience replay buffer provides independent environment samples to the critic and actor networks; such a buffer is necessary because neural networks require i.i.d samples to train effectively [30]. The experience replay buffer requires the algorithm to be off-policy, as the training updates are carried out on samples collected from a random historic policy rather than the one being updated[24]. Lastly, the frozen target networks from the DQN algorithm were employed to help handle the non-stationarity caused by the moving values of the actor and the critic between updates. However, aside from these features, the core of DDPG is simply the off-policy deterministic actor-critic method [24]; indeed, the algorithm's pseudocode, given below, demonstrates this point.

Algorithm 1 DDPG algorithm

Randomly initialise critic $Q(s,a \mid \theta^Q)$ and actor $\mu(s \mid \theta^\mu)$ with weights θ^Q and θ^μ
Initialise target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialise replay buffer R
for episode = $1, M$ **do**
    Initialise a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for $t = 1, T$ **do**
        Select action $a_t = \mu(s_t \mid \theta^\mu) + \mathcal{N}_t$ according policy and exploration noise
        Execute action a_t and observe reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$
        Update critic by minimising the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,a \mid \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s \mid \theta^\mu)|_{s_i}$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for

Importantly, Lillicrap et al. [24] make the assumption for continuous action spaces that the deterministic action selection is differentiable. The same does not hold true

on discrete action spaces. However, DDPG can train on discrete action spaces through the use of a Gumbel-Softmax [21] to ensure the deterministic action selection can be backpropagated through during training (e.g. [26]).

This section has described the DDPG algorithm and explained its advantages over other RL algorithms in high-dimensional, continuous action space environments. The following section concludes the background chapter by introducing multi-agent reinforcement learning.

2.6 Multi-Agent Reinforcement Learning

The topics discussed so far in the background chapter have considered single-agent RL. However, as is the project's focus, we now turn our attention to multi-agent RL (MARL), a method whereby multiple agents repeatedly interact with the same environment to learn to solve a given multi-agent task (e.g. [34]). Since the previously introduced MDP framework is explicitly defined for the single-agent RL case, we now consider an alternate framework for MARL.

Partially-observable stochastic games (POSGs) for N agents (e.g. [18]) define seven important elements. The first four of which hold information on the state of the environment and the agents. They are given by: $\mathcal{N} = \{1,\dots,N\}$, the set of all agents; $\mathcal{S}$, the state space describing the possible arrangements of all agents; the joint observation space $\mathcal{O} = O^1 \times \cdots \times O^N$; and lastly, the joint action space $\mathcal{A} = A^1 \times \cdots \times A^N$.

The final three elements allow POSGs to provide new environment transitions. Each is a function mapping from a subset of the previously observed information and are given formally here: $\Omega : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{O})$, the observation function, it defines the local observations for each agent $o^i \in O^i$; $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ defines a distribution over next states given the current state and joint actions. Finally, $\mathcal{R}^i$ defines the reward function for all agents i and is given by $\mathcal{R}^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$.

The learning objective in POSGs is to find policies $\boldsymbol{\pi} = (\pi_1, \dots, \pi_N)$ for all N agents, such that the discounted return of each agent i, $R^i = \sum_{t=0}^{T} \gamma^t r_t^i$, is maximised with respect to the other policies in $\boldsymbol{\pi}$, where $r_t^i \in \mathcal{R}^i$ is the individual agent reward, γ is the discount factor and T is the maximum timestep of an episode. We write this as: $\forall_i : \pi_i \in \arg\max_{\pi_i'} \mathbb{E}[R^i \mid \pi_i', \boldsymbol{\pi}_{-i}]$, where $\boldsymbol{\pi}_{-i} = \boldsymbol{\pi} \setminus \{\pi_i\}$.

Chapter 3

Related Work

Centralised and decentralised methods refer to the information passed to agents at training and execution time. A fully centralised method trains and executes agents on global information. In contrast, a fully decentralised method trains and executes agents solely on local information. Notably, most MARL methods employ either a fully centralised or fully decentralised approach [19]. Unfortunately, these two methods can sometimes fall short. For example, complete centralisation, which conditions each agent's policy on the joint action space, is not always practical for tasks with large action spaces or many agents [13]. Furthermore, decentralised methods often fail to learn complex behaviour between agents, such as cooperation [35].

Consequently, researchers have introduced the centralised training and decentralised execution (CTDE) method to fill the gap [15, 22]. In brief terms, CTDE utilises global information during training—made feasible in real-world applications with simulators where agents can communicate freely [13]—while executing agent policies conditioned solely on local information.

The algorithm that we propose utilises the CTDE method. Therefore, this section will examine various other CTDE algorithms that already exist within the literature to provide context that helps understand how our findings add to the current state of MARL research. We also briefly outline a decentralised method utilised as a baseline within our research for the same reason.

3.1 Independent Actor-Critic

Independent Actor-Critic (IAC) algorithms are decentralised forms of MARL. First introduced by Tan [46] with Independent Q-learning, they are often used as baselines

in research papers (e.g. [13, 52]) as they provide a naive approach to the MARL problem, and yet frequently prove to be effective on tasks with fewer agents and fewer complexities (such as required agent cooperation) in the environment [35, 29]. As an example, we focus on the Independent synchronous Advantage Actor-Critic algorithm, a variant on A2C [31, 12].

3.1.1 Independent synchronous Advantage Actor-Critic (IA2C)

IA2C operates multiple actors within an environment. Each actor's policy is updated with a critic, which receives local observations—identical to A2C [31]—forcing the agents to operate and act independently [12]. As with the A2C algorithm, IA2C utilises multiple parallel environments to gather i.i.d transition samples. Additionally, each agent in IA2C minimises the A2C loss [35]. Our proposed algorithm utilises CTDE, a method shown to surpass independent algorithms, in terms of both maximum and average returns, in environments that require coordination between agents.

3.2 Centralised Policy Gradient

As mentioned, several CTDE algorithms already exist within the MARL literature. These include Multi-Agent DDPG [26], Multi-Agent A2C (e.g. [35]), Multi-Agent Proximal Policy Optimisation [52], and Counterfactual Multi-Agent [13]. We describe each, examining their advantages and disadvantages for different MARL environments [35]. Furthermore, we also discuss how our proposed approach looks to resolve some of the issues faced by the current algorithms.

3.2.1 Multi-Agent DDPG (MADDPG)

Within their paper, Lowe et al. [26] introduce the Multi-Agent Actor-Critic (MAAC) method. In addition, they also propose an original algorithm, called Multi-Agent DDPG (MADDPG), that combines the single-agent DDPG algorithm with their MAAC method.

Precisely, MAAC is a CTDE method [26], where the critic is conditioned on the joint state and action, and the actor is conditioned on local observations. Formally, the critic is written: $Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N)$, where $\mathbf{x} = (o_1, \ldots, o_N) \in \mathcal{O}$ is the joint observation, and $a_1, \ldots a_N$ the joint action, and the outputs are the joint-state value for agent $i \in \mathcal{N}$.

Furthermore, each $Q_i^{\boldsymbol{\pi}}$ is trained separately allowing agents to learn conflicting rewards in competitive settings [26].

The objective function is the expected return for each agent $J(\theta_i) = \mathbb{E}[R_i]$, and we write the gradient estimate for the objective function as follows:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a_i\sim\pi_i}[\nabla_{\theta_i} \log \pi_i(a_i \mid o_i) Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, \ldots, a_N)], \tag{3.1}$$

MADDPG extends MAAC to work for deterministic policies. That is, each agent minimises the DDPG loss [24]. We therefore write the gradient estimate of the MADDPG objective function as:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{\mathbf{x}, a\sim\mathcal{D}}[\nabla_{\theta_i} \mu_i(a_i \mid o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N)|_{a_i=\mu_i(o_i)}], \tag{3.2}$$

where $\boldsymbol{\mu}$ represents the deterministic policies for all N agents, parameterised with respect to θ_i for each agent i.

Notably, Lowe et al. [26] test the MADDPG algorithm on discrete action space environments, where the assumption that the deterministic actions are differentiable with respect to the policy parameter does not apply [24]. Hence, Lowe et al. [26] employ a Gumbel-Softmax to ensure that MADDPG's actions remain differentiable in discrete action spaces [21].

Following on, Lowe et al. [26] demonstrate that MADDPG provides an advantage over independent methods in environments where complex coordination between agents is necessary [32]. Papoudakis et al. [35] found similar results for MADDPG in the same Multi-agent Particle Environments (MPEs) used by Lowe et al. [26]. However, they also found that MADDPG achieved far lower average and maximum returns when tested on other discrete action environments (e.g. [1, 36]); even when compared to independent algorithms. Papoudakis et al. [35] suggest that the biased categorical reparameterisation introduced by Gumbel-Softmax could be causing the poor returns observed in the discrete action environments.

Our proposed algorithm removes the need for a Gumbel-Softmax, by combining the MAAC method with a stochastic policy gradient algorithm, instead of DDPG. Thus, comparing our algorithm against MADDPG on discrete action environments should provide insight into how Gumbel-Softmax affects the observed returns for MADDPG.

3.2.2 Multi-Agent A2C (MAA2C)

MAA2C is a CTDE variant of the IA2C algorithm. As with IA2C, policy updates are conditioned on the actions taken by the individual agents, and baselined with the advantage function. However, unlike IA2C, MAA2C applies a centralised critic conditioned on the joint state. In particular, this also differs from MADDPG where the critic is conditioned on both the joint state and joint action.

Explicitly, the objective function for MAA2C is the expected return for each agent, $J(\theta_i) = \mathbb{E}[R_i]$, and the gradient estimate is given by:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a_i\sim\pi_i}[\nabla_{\theta_i} \log \pi_i(a_i \mid o_i) A_w(\mathbf{x}, a_i)], \tag{3.3}$$

where $A_w(\mathbf{x}, a_i) = R_i + \gamma V_w(\mathbf{x}') - V_w(\mathbf{x})$, and $w \in \mathbb{R}^d$ are the state-value weights learnt by the critic during training.

Regarding the algorithms average and maximum returns, Papoudakis et al. [35] found that MAA2C performs competitively against other MARL algorithms in all except the challenging StarCraft Multi-Agent Challenge (SMAC) [36]. These findings for MAA2C on SMAC are also supported by Vasilev et al. [47], while utilising MAA2C as a baseline in their experiments. Moreover, given that the main difference between IA2C and MAA2C is the centralised critic condition.

Importantly, MAA2C can not be used to answer our research question because it conditions the critic solely on the joint observation, and therefore, does not employ the MAAC method utilised by MADDPG [26]. Thus, our work differs from MAA2C because we will condition the critic on both the joint state and joint action, providing a clear comparison to determine Gumbel-Softmax's impact on MADDPG within discrete action environments.

3.2.3 Multi-Agent PPO (MAPPO)

MAPPO is a CTDE extension of PPO [52]. Specifically, the critics in MAPPO learn a centralised state value function, while the actors' policies are conditioned on the local observations and are updated by maximising the PPO objective function (2.11).

$$L(\theta) = \frac{1}{B_n} \sum_{i=1}^{B} \sum_{k=1}^{n} \min\left(r_{\theta,i}^{(k)} A_i^{(k)}, \text{clip}(r_{\theta,i}^{(k)}, 1-\varepsilon, 1+\varepsilon) A_i^{(k)}\right) + \xi, \tag{3.4}$$

where B refers to the batch size, n the number of agents, $r_{\theta,i}^{(k)} = \frac{\pi_\theta(a_i^{(k)}|o_i^{(k)})}{\pi_{\theta_{\text{old}}}(a_i^{(k)}|o_i^{(k)})}$, and $A_i^{(k)}$ is the GAE method, computed using (2.12) with the state value function replaced with

the joint state value. Additionally, ξ is the sum over the each agents' policy entropy averaged per batch and is given as follows:

$$\xi = \sigma \frac{1}{B_n} \sum_{i=1}^{B} \sum_{k=1}^{n} S[\pi_\theta(o_i^k)], \tag{3.5}$$

where S is the policy entropy and σ is the entropy coefficient. Maximising this term within each actor's policy update ensures that less frequently visited states are explored [50].

Yu et al. [52] found that MAPPO achieves comparable maximum returns to various off-policy algorithms on discrete cooperative multi-agent environments. Moreover, Yu et al. [52] also found that MAPPO achieves a sample efficiency similar to the off-policy algorithms they compared against, which is remarkable considering that usually on-policy algorithms are far less sample efficient [52]. In their benchmarking paper, Papoudakis et al. [35] obtain results that support these findings, concluding that MAPPO's main advantage over similar algorithms (such as MAA2C) is its combination of on-policy optimisation with its surrogate objective function.

As mentioned, MAPPO conditions the critic on the joint state, in contrast to MADDPG, which conditions its critic on both the joint state and joint action. Hence, MAPPO can not provide a clear comparison to MADDPG for determining the impact of Gumbel-Softmax. Therefore, our proposed algorithm differs from MAPPO because it will instead condition the critic on both the joint state and joint action to determine Gumbel-Softmax's impact on MADDPG within discrete action environments [26].

3.2.4 Counterfactural Multi-Agent (COMA)

Foerster et al. [13] present the COMA algorithm to improve decentralised policy gradient estimates under a centralised critic. In order to achieve this, the authors apply a baseline to the policy update inspired by difference rewards [51]. We now detail COMA and discuss its performance under several metrics as measured by Foerster et al. [13] and compare it to the measurements made by Papoudakis et al. [35] in their MARL benchmarking paper, and those made within a paper which proposes improvements to COMA [47].

Similarly to MADDPG, COMA employs a centralised critic conditioned on the joint observation and joint action while preserving independent policies for each agent [26, 13]. However, unlike MADDPG, the COMA algorithm utilises a counterfactual baseline on the actors' policy updates, enabling credit assignment per agent [13]—the

policy update can then track how much the individual agent's actions contributed to the global reward. Furthermore, the paper's authors propose a specific model architecture that uses a single central critic for all agents to enable a more efficient calculation of the counterfactual baseline [13]. The counterfactual baseline proposed by Foerster et al. [13] for each agent i is as follows:

$$A^i(\mathbf{x},\mathbf{u}) = Q(\mathbf{x},\mathbf{u}) - \sum_{a_i^*} \pi_i(a_i^* \mid o_i) Q(\mathbf{x},(\mathbf{u}_{-i},a_i^*)) \tag{3.6}$$

where $\mathbf{u}$ is the joint action, $\mathbf{u}_{-i}$ is the joint action not including the action of agent i, and u_*^i is a possible action for agent i. The objective function for proposed by Foerster et al. [13] can then be given as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a_i\sim\pi_i}[\nabla_{\theta_i} \log \pi_i(a_i \mid o_i) A_w^i(\mathbf{x},\mathbf{u})] \tag{3.7}$$

Foerster et al. [13] compared COMA to baselines (including MAA2C, MAPPO and IA2C) on the StarCraft unit micromanagement environment and found that it successfully outperforms them in terms of the average number of game wins during training. However, their findings were contradicted by those in the benchmarking paper [35], where COMA was found to perform worse in terms of average and maximum returns on a wide variety of multi-agent environments, including in the SMAC environment [36]. Furthermore, Vasilev et al. [47] also found that COMA performed poorly on the SMAC tasks against several of the same baselines mentioned.

COMA does not answer the research questions we set out for this project because of its many differences with MADDPG. Despite being a CTDE algorithm, COMA's use of a single centralised critic prevents it from learning on competitive environments. Our proposed algorithm will utilise the MAAC method, therefore making it a far more useful comparison to MADDPG for answering our questions around Gumbel-Softmax.

Chapter 4

Methods

The project's primary objective is to compare a stochastic RL algorithm that utilises the MAAC framework against MADDPG to experimentally determine whether the reparametrisation bias introduced by Gumbel-Softmax [21, 35], required by MADDPG to learn on discrete action spaces [26], impacts its 'performance'. As detailed later in this chapter, the 'performance' of each algorithm will be measured using several metrics, including the maximum evaluation returns, average evaluation returns, and average compute time. Furthermore, we will test for these metrics on multi-agent environments with different properties (such as sparse rewards vs non-sparse rewards, cooperation vs non-cooperation, different levels of complexity) to discover whether the use of a deterministic policy with a Gumbel-Softmax has any trade-offs in discrete action spaces [26, 24]. Or if stochastic MAAC algorithms, which do not require a Gumbel-Softmax, always perform better in terms of the metrics we are measuring.

4.1 Baseline Algorithms

4.1.1 IA2C

The importance of employing IA2C as a baseline is two-fold. Firstly, it will allow us to determine where the use of centralised learning becomes advantageous over decentralised learning on a given task. Secondly, it provides a clear data point on how on-policy algorithms should perform, which is essential because the algorithm we introduce is on-policy.

4.1.2 MADDPG

Comparing MADDPG with our proposed algorithm is essential to the project. It will provide the experimental data required to answer our research question around whether Gumbel-Softmax impedes the performance, as measured under our metrics, of MAADPG on discrete action environments. Furthermore, using MADDPG as a baseline will also provide insights into off-policy vs on-policy learning with the MAAC method since our proposed algorithm is on-policy.

4.1.3 MAPPO

With the MAPPO baseline, it will be possible to determine whether our proposed algorithm performs at or above a level already present in the CTDE MARL literature [52, 35]. Moreover, MAPPO will also enable a direct comparison between joint state value critics and joint state-action value critics, a property not represented in the other baselines.

4.2 Proposed Algorithm

4.2.1 Multi-Agent PPO Q-learning (MAPPOQ)

We propose MAPPOQ to compare against MADDPG and empirically determine the impact of the reparametrisation bias introduced by its Gumbel-Softmax [35]. Below we outline our proposed algorithm in further detail.

MAPPOQ is an on-policy and stochastic MAAC algorithm. Therefore, like MADDPG, actors in MAPPOQ learn from their local observation histories while critics are conditioned on the joint observations and actions; each critic within MAPPOQ approximates the joint state-action value function. Additionally, MAPPOQ applies various techniques from MADDPG and IA2C. For example, target networks are implemented to help stabilise the critic's update during training, as seen in MADDPG. Furthermore, MAPPOQ trains agents in parallel environments, like IA2C, to ensure that i.i.d samples are collected despite the algorithm being on-policy [31].

In contrast to MADDPG and IA2C, MAPPOQ minimises the PPO objective function. We expect PPO to allow MAPPOQ to be an appropriate comparison to the off-policy MADDPG baseline when compared for sample efficiency, given the reuse of initial on-policy samples to update the PPO surrogate objective function [38, 52]. Also

in contrast to MADDPG and IA2C, MAPPOQ employs RNNs. We decided upon this to remain consistent with the MAPPO implementation. Below we provide the pseudo-code for our proposed MAPPOQ algorithm.

Algorithm 2 MAPPOQ algorithm

Initialise policy and critic weights, θ^{π} and θ^{Q}, using Orthogonal initialisation [20]
Initialise target networks π' and Q' with weights $\theta^{\pi'} \leftarrow \theta^{\pi}, \theta^{Q'} \leftarrow \theta^{Q}$
Set learning rates $\alpha^{\pi}, \alpha^{Q} > 0$
while $step \leq step_{max}$ **do**
    Set data buffer $D = \{\}$
    for $j = 1$ **to** $batch\_size$ **do**
        $\mathcal{T} = []$ empty list
        Initialise $h_{0,\pi}^{(1)} \ldots, h_{0,\pi}^{(n)}$, $h_{0,Q}^{(1)} \ldots, h_{0,Q}^{(n)}$ RNN states
        for $t = 1$ **to** T **do**
            $\mathbf{u_t}$ = [] empty list
            for all agents i **do**
                $p_t^{(i)}, h_{t,\pi}^{(i)} = \pi(o_t^{(i)}, h_{t-1,\pi}^{(i)} \mid \theta_i^{\pi})$
                $u_t^{(i)} \sim p_t^{(i)}$
                Append $u_t^{(i)}$ to list $\mathbf{u_t}$
            end for
            for all agents i **do**
                $v_t^{(i)}, h_{t,Q}^{(i)} = Q(\mathbf{x_t}, \mathbf{u_t}, h_{t-1,Q}^{(i)} \mid \theta_i^{Q})$
            end for
            Execute actions $\mathbf{u_t}$, observe $r_t, \mathbf{o_{t+1}}, \mathbf{x_{t+1}}$
            $\mathcal{T}$ += $[\mathbf{o_t}, \mathbf{x_t}, \mathbf{h_{t,\pi}}, \mathbf{h_{t,Q}}, \mathbf{u_t}, r_t, \mathbf{o_{t+1}}, \mathbf{x_{t+1}}]$
        end for
        Predict next actions $\mathbf{u_{T+1}} \sim \mathbf{p_{T+1}}$
        Compute discounted returns R on $\mathcal{T}$
        Divide trajectory $\mathcal{T}$ into chunks of length L and append to D
    end for
    Adam update θ^{Q} on $L(\theta^{Q})$ and θ^{π} on $L(\theta^{\pi})$ with data D
    if $step = update\_freq$ **then**
        Update target networks $\theta^{\pi'} \leftarrow \theta^{\pi}, \theta^{Q'} \leftarrow \theta^{Q}$
    end if
end while

where $L(\theta_i^Q)$ represents the critic loss for each agent and is given by:

$$L(\theta_i^Q) = \frac{1}{T}\sum_t \left(R_t - Q\left(\mathbf{x_t}, \mathbf{u_t}, h^{(i)}_{t-1,Q} \mid \theta_i^Q\right)\right)^2, \tag{4.1}$$

where R_t is the discounted returns for timestep t. $L(\theta_i^\pi)$ is the actor loss for each agent and is given by:

$$L(\theta_i^\pi) = \frac{1}{T}\sum_t \min\left(r_{\theta,t}R_t, \text{clip}(r_{\theta,t}, 1-\varepsilon, 1+\varepsilon)R_t\right) + \xi, \tag{4.2}$$

where $r_{\theta,i}$ is the PPO importance sample between the old and new policies shown after equation (3.4), and ξ is the policy entropy term similar to equation (3.5).

4.3 Environments

4.3.1 Level-Based Foraging (LBF)

LBF is a multi-agent grid-world environment where agents collect randomly placed apples to receive rewards [1, 9]. Agents and apples are assigned levels. Where an agent's level is less than or equal to the level of an apple, they may collect it. Otherwise, enough agents must cooperate such that their combined levels are greater than or equal to the level of the apple.

The LBF environment allows for varying levels of complexity (such as smaller or larger grids, fewer or more agents, fewer or more apples, full or partial observability), enabling experiments to stretch the algorithms and provide a great range of insights into their abilities; including how they handle the complexities mentioned and how they handle cooperation when required. Furthermore, the LBF environment has sparse rewards, which can cause issues with exploration during learning [48]. Thus, it provides the opportunity to examine the algorithms under this particular constraint.

Accordingly, we run experiments across seven LBF environment settings, where we vary the observability, grid size, number of agents, and number of apples. These environments are given by: (1): 8x8-2p-2s; (2) 2s-8x8-2p-2f; (3) 8x8-2p-3f; (4) 10x10-2p-2f; (5) 2s-10x10-2p-2f; (6) 10x10-2p-3f; (7) 15x15-3p-4f. Where the 2s refers to partial observability. The following values then represent the grid size, number of agents, and number of apples respectively.

These LBF experiments were selected as they provide an increasing range of difficulties. Furthermore, as the difficulties increase, we expect that the algorithms will be required to learn more complex cooperative behaviour between agents; typically

thought to favour CTDE algorithms and therefore provides the perfect setting to measure differences between MADDPG and our proposed algorithm.

4.3.2 Multi-Agent Particle Environments (MPEs)

MPEs consists of different tasks involving multiple agents and landmarks [32, 26]. In some of the tasks, agents must cooperate to attain rewards (Spread and Speaker Listener), and in others, agents must compete to achieve rewards (tag and adversary). The tasks set out in MPEs require high levels of cooperation amongst agents, requiring algorithms to learn complex and coordinated policies [35, 26].

An advantage of using MPEs is that they provide a perfect comparison to the original MAADPG paper because the authors experimented on MPE tasks exclusively [26]. Another advantage comes from the high levels of coordination required to solve the tasks. Previous research has demonstrated that this requirement allows CTDE algorithms to outperform their independent counterparts [35, 26]. Thus, MPEs will enable a clear benchmark for our proposed algorithm.

We report evaluation rewards on the following MPE tasks: (1) Speaker Listener (or Cooperative communication), a task in which a stationary agent must communicate the location of a target to a non-stationary agent, whose goal is to reach the said target; (2) Spread (or Cooperative navigation), where agents are required to learn to keep close to a set of landmarks while simultaneously avoiding collisions with the other agents in the world.; (3) Tag (or Predator-prey), which involves three adversary agents and one good agent. The good agent is faster and must avoid being hit by the adversarial agents; Lastly, (4) Adversary (or Physical deception), a task involving two landmarks (one a target and one not), two good agents who know where the target landmark is, and one adversary who does not. All agents are rewarded for how close they are to the target. However, the good agents are negatively rewarded if the adversary is close to the target. Hence, the good agents must learn to deceive the adversary by splitting up and covering both the target and non-target landmarks.

4.4 Performance Metrics

4.4.1 Maximum Returns

We report each algorithm's highest average return value from the intermediary evaluation timesteps during training. The value reported will be the average across five

random seeds from the timestep. A confidence interval of ninety-five per cent will also be reported for each maximum return.

4.4.2 Average Returns

Averaging the values from each evaluation timestep during training provides a metric that accounts for the convergence speed/sample efficiency and the maximum return achieved. As with the maximum return metric, we average the mean evaluation across five random seeds and report the ninety-five per cent confidence intervals around the value.

4.4.3 Compute Time

For further insight, we also measure the wall-clock time taken by each algorithm to complete training. This performance metric is necessary because a model may have excellent sample efficiency and yet take far more time to train than other algorithms; this is especially true when comparing on-policy and off-policy algorithms [35].

4.5 Evaluation Protocol

We train each algorithm for three million timesteps on the environments, evaluating ten episodes every thirty-one thousand steps for a total of ninety-seven evaluations per training run. We record the average evaluation rewards across five random seeds, along with the standard deviation, for each evaluation step. Keeping the evaluation protocol consistent in this way will help draw quick conclusions on our results, particularly the average returns and compute time. As a side note, three million training timesteps were chosen based on results found during hyperparameter tuning and those mentioned in the benchmarking paper by Papoudakis et al. [35].

Furthermore, we report the ninety-five per cent confidence interval for each mean value across the five seeds. To do this, we utilise the sample standard deviation and the t-distribution. Reporting the ninety-five per cent confidence interval allows us to assess the variance of each method easily.

4.6 Hyperparameter Optimisation

If optimal hyperparameters were published, we typically utilised them (e.g. [35, 52]). However, when no hyperparameter information was available, we performed a grid-search on one task per environment, retaining the hyperparameters that lead to the best results across each of our performance metrics.

4.7 Implementation Details

All experiments were conducted with the Microsoft Azure cloud computing service [1]. The machines used for evaluation were equipped with four virtual CPUs, 16GiB RAM, and 32GiB temporary storage.

The software we developed made use of several publicly available PyTorch implementations of the Actor-Critic algorithms [2] and MADDPG [3].

4.8 Difficulties

Before arriving at the MAPPOQ algorithm we spent a significant amount of time attempting to implement the MAAC method with the A2C algorithm. Unfortunately, MAACQ suffers from poor sample efficiency, and gave little insight into the impact of Gumbel-Softmax on MADDPG's performance. Following this, we abandoned MAACQ and implemented a more sample efficient algorithm, PPO [38].

[1] `https://azure.microsoft.com`
[2] `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`
[3] `https://github.com/shariqiqbal2810/maddpg-pytorch`

Chapter 5

Results

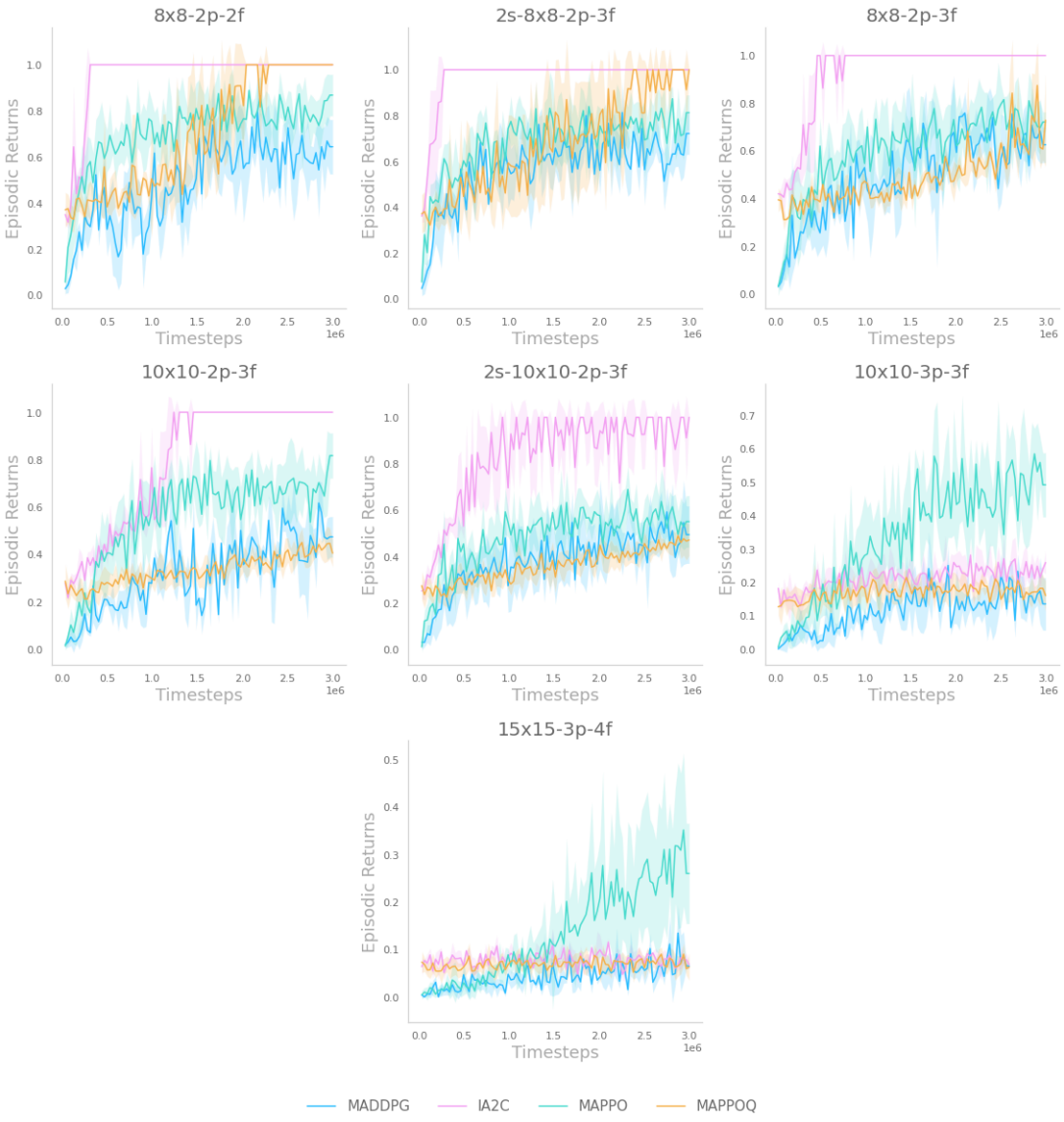

Figure 5.1: Visual plots from the seven LBF tasks.

5.1 LBF

5.1.1 Maximum Returns

Table 5.1: Maximum evaluation returns and 95% confidence interval reported over five training seeds for each algorithm on the LBF tasks. Returns in bold represent the highest score for that task.

Task	IA2C	MADDPG	MAPPO	MAPPOQ
8x8-2p-2f	$\mathbf{1.00 \pm 0.00}$	0.79 ± 0.19	0.89 ± 0.06	$\mathbf{1.00 \pm 0.00}$
2s-8x8-2p-2f	$\mathbf{1.00 \pm 0.00}$	0.81 ± 0.16	0.87 ± 0.07	$\mathbf{1.00 \pm 0.00}$
8x8-2p-3f	$\mathbf{1.00 \pm 0.00}$	0.76 ± 0.11	0.82 ± 0.02	0.87 ± 0.22
10x10-2p-3f	$\mathbf{1.00 \pm 0.00}$	0.62 ± 0.12	0.82 ± 0.12	0.45 ± 0.08
2s-10x10-2p-3f	$\mathbf{1.00 \pm 0.00}$	0.59 ± 0.13	0.69 ± 0.12	0.49 ± 0.05
10x10-3p-3f	0.28 ± 0.03	0.25 ± 0.07	$\mathbf{0.59 \pm 0.11}$	0.21 ± 0.03
15x15-3p-4f	0.12 ± 0.01	0.13 ± 0.07	$\mathbf{0.35 \pm 0.20}$	0.09 ± 0.03

MAPPOQ achieved higher maximum returns than both MAPPO and MADDPG within the eight-by-eight grids. Hence, the hypothesis that Gumbel-Softmax hinders MADDPGs ability to approximate the optimal policy is supported. However, for tasks with larger grid sizes, the maximum returns attained by MAPPOQ drops below those of both MADDPG and MAPPO. We, therefore, infer that MAPPOQ does not handle sparse reward tasks well compared to MADDPG and MAPPO since increased grid sizes in LBF reduce the state vs reward ratio.

Interestingly, comparing results from the 10x10-2p-3f and the 2s-10x10-2p-3f tasks, we find the maximum returns achieved by MAPPOQ are slightly higher in the latter. It suggests that the centralised critic within MAPPOQ carries extra benefit in environments where the actor only has partial observability over those where the actor fully observes the environment. However, the evidence for this implication is unfortunately weak, given that the confidence intervals between the two values heavily overlap. Moreover, the results do not provide indications that this applies more generally to all CTDE algorithms, given that the returns of MADDPG and MAPPO both tend to fall when the partial observability condition is applied.

Finally, observing the results for IA2C, we see that it achieved the highest maximum possible returns for LBF across all seeds in five of the seven tasks. Furthermore, comparing its results with the other algorithms suggests that for the LBF tasks we experimented on, CTDE has no advantage over complete decentralisation—this is all

observed in the average return results that we discuss next. Therefore, the implication is that CTDE methods do not provide the perfect solution to every MARL task. A finding that has been found noted before by Lyu et al. [27]. In the discussion chapter, we shall detail our hypothesis for why this is the case.

5.1.2 Average Returns

Table 5.2: Average evaluation returns and 95% confidence interval reported over five training seeds for each algorithm on the LBF tasks. Returns in bold represent the highest score for that task.

Task	IA2C	MADDPG	MAPPO	MAPPOQ
8x8-2p-2f	$\mathbf{0.96 \pm 0.06}$	0.48 ± 0.16	0.71 ± 0.15	0.70 ± 0.16
2s-8x8-2p-2f	$\mathbf{0.97 \pm 0.06}$	0.57 ± 0.13	0.68 ± 0.14	0.68 ± 0.22
8x8-2p-3f	$\mathbf{0.93 \pm 0.07}$	0.50 ± 0.16	0.60 ± 0.16	0.49 ± 0.09
10x10-2p-3f	$\mathbf{0.79 \pm 0.08}$	0.31 ± 0.13	0.56 ± 0.15	0.33 ± 0.06
2s-10x10-2p-3f	$\mathbf{0.83 \pm 0.15}$	0.38 ± 0.14	0.48 ± 0.13	0.36 ± 0.07
10x10-3p-3f	0.21 ± 0.05	0.12 ± 0.07	$\mathbf{0.34 \pm 0.15}$	0.17 ± 0.05
15x15-3p-4f	0.07 ± 0.02	0.04 ± 0.03	$\mathbf{0.13 \pm 0.11}$	0.07 ± 0.02

Similar patterns found with the maximum returns also appear in the average returns. Additionally, the average return metric enables inferences on the algorithms' sample efficiencies; we now examine these details.

The average returns on LBF show that MAPPOQ's learning rate falls when increasing the grid sizes, supporting the previous suggestion that MAPPOQ struggles to learn optimal policies in sparsely rewarded tasks. Additionally, MAPPOQ achieved higher average returns than MADDPG within the smaller grid sizes, supporting the hypothesis that Gumbel-Softmax impedes MADDPG in discrete action space environments.

Analysing the results between the partially observable states and their fully observable counterparts, we notice that the seeming effect of improved performance mentioned in the maximum returns section has vanished. Hence, the evidence for the previously mentioned implication is not supported under this metric.

The implication that CTDE algorithms are unnecessary for the LBF tasks we measured remains firmly supported, as IA2C outperforms all the CTDE algorithms for average returns. Furthermore, both IA2C and MAPPOQ use stochastic policies, which are advantageous in sparse reward environments given the inherent exploration from

sampling the stochastic policy. However, despite MAPPOQ sharing this feature, it did not achieve the same average returns as IA2C.

Lastly, the sample efficiency of MAPPOQ is comparable to MADDPG. We hypothesised that this would be the case, given that our algorithm uses the PPO surrogate objective function, a method that was shown to improve on-policy sample efficiency by Yu et al. [52].

5.1.3 Compute Time

Table 5.3: Average wall-clock time in seconds and 95% confidence interval reported over five training seeds for each algorithm on the LBF tasks. Times in bold represent the fastest time for that task.

Task	IA2C	MADDPG	MAPPO	MAPPOQ
8x8-2p-2f	**401.12 ± 2.02**	3577.41 ± 21.63	6050.91 ± 139.15	4308.98 ± 80.36
2s-8x8-2p-2f	**401.12 ± 2.02**	3611.52 ± 24.26	5735.52 ± 163.87	4045.07 ± 20.63
8x8-2p-3f	**393.63 ± 0.48**	3506.64 ± 10.23	5770.55 ± 50.03	4301.53 ± 99.07
10x10-2p-3f	**398.86 ± 0.74**	4580.40 ± 161.52	5774.35 ± 39.17	5689.34 ± 322.74
2s-10x10-2p-3f	**390.31 ± 7.66**	3574.32 ± 46.64	5788.85 ± 71.35	5731.40 ± 252.07
10x10-3p-3f	**550.35 ± 3.31**	6780.87 ± 1890.47	8543.86 ± 37.45	7839.43 ± 494.56
15x15-3p-4f	**543.06 ± 3.98**	8039.77 ± 175.46	8605.00 ± 55.80	8302.61 ± 368.24

All four algorithms' compute time increased as the number of agents increased. Additionally, IA2C was far faster than the CTDE algorithms; this suggests that CTDE inclusion of extra information within the critic slows down the updates. However, since the PPO algorithms iterate many times over their update function, it is expected that these algorithms would be slower. Lastly, MAPPOQ achieved similar times to MADDPG. Therefore, we infer that MAAC implemented with the stochastic policy of PPO is equal in computational cost to MAAC implemented with DDPG and Gumbel-Softmax.

5.2 MPEs

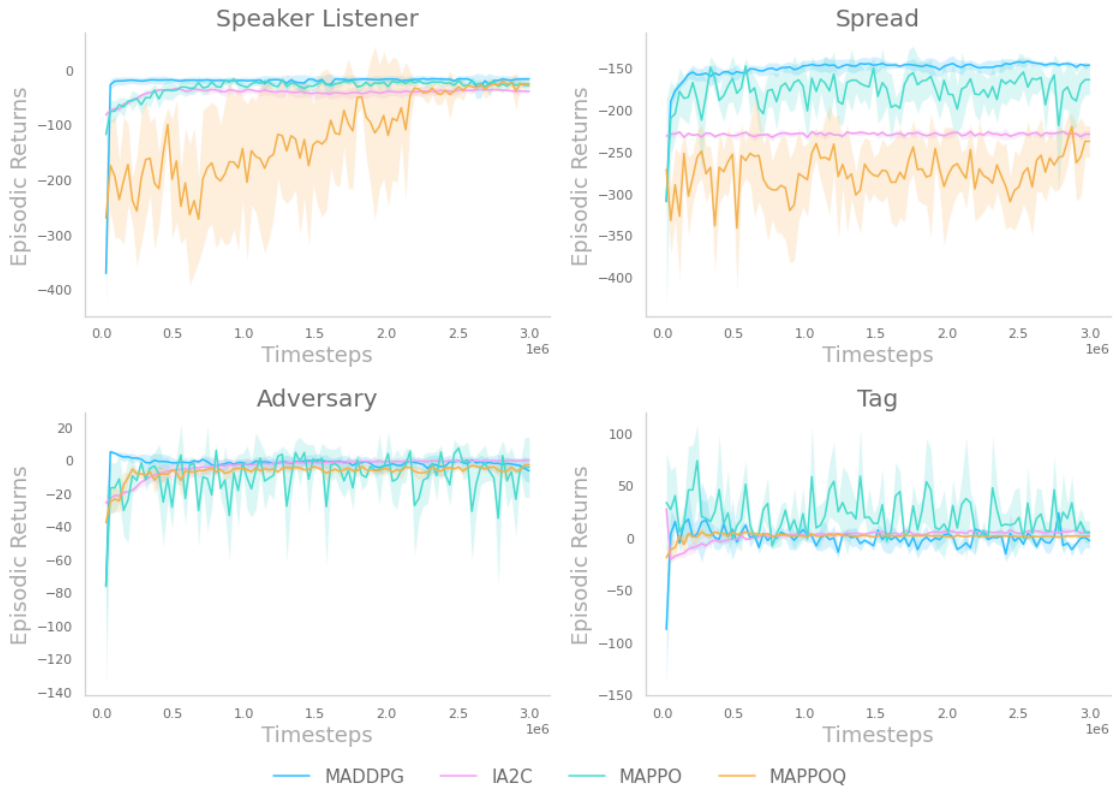

Figure 5.2: Visual plots from the MPE tasks.

5.2.1 Maximum Returns

Table 5.4: Maximum evaluation returns and 95% confidence interval reported over five training seeds for each algorithm on the MPE tasks. Returns in bold represent the highest score for that task.

Task	IA2C	MADDPG	MAPPO	MAPPOQ
Speaker Listener	-33.89 ± 2.60	$\mathbf{-16.17 \pm 6.78}$	-16.93 ± 5.81	-20.44 ± 8.30
Spread	-226.25 ± 5.88	$\mathbf{-142.40 \pm 6.26}$	-148.04 ± 10.81	-220.15 ± 19.97
Adversary	-0.21 ± 1.72	$\mathbf{4.81 \pm 2.00}$	-1.94 ± 1.68	-3.01 ± 1.72
Tag	$\mathbf{27.30 \pm 2.49}$	23.36 ± 9.66	10.23 ± 3.37	5.27 ± 3.22

Within Speaker listener and Spread, the two cooperative MPE tasks, MAPPOQ, achieved lower returns than MAPPO and MADDPG. Therefore, suggesting that the algorithm could not learn as successful a policy compared to the other two algorithms.

However, despite not learning as well as MADDPG and MAPPO, our proposed algorithm successfully attained a much higher score than IA2C. As a fully decentralised algorithm, it is not surprising that IA2C fails to learn the cooperative behaviour necessary for solving the Speaker Listener task given the lack of consistent gradient signal [26]. It suggests that MAPPOQ's centralised critic enables it to learn the required cooperative behaviour, as we would hope to find. On the other hand, MAPPOQ suffered from a far lower return for the spread environment and displayed the highest variance out of the four algorithms. Given that Spread involves three agents, compared to the two within the Speaker Listener environment, it suggests that MAPPOQ struggles to learn in environments with many agents.

Continuing to the competitive environments, we observe that MAPPOQ attained the lowest score for both tasks out of all the algorithms. However, this does not suggest that MAPPOQ struggled with these environments because successfully solving the competitive tasks involves learning a stable policy where the agents find equilibrium. Hence, a successful learning curve within the competitive tasks should show stable learning behaviour without sudden significant increases or decreases in returns. Therefore, we wait until the average returns section before commenting on the performances observed within these competitive environments.

5.2.2 Average Returns

Table 5.5: Average evaluation return and 95% confidence interval reported over five training seeds for each algorithm on the MPE tasks. Returns in bold represent the highest score for that task.

Task	**IA2C**	**MADDPG**	**MAPPO**	**MAPPOQ**
Speaker Listener	-41.50 ± 7.89	$\mathbf{-22.95 \pm 12.74}$	-31.34 ± 12.47	-120.94 ± 114.06
Spread	-229.84 ± 4.10	$\mathbf{-151.93 \pm 8.82}$	-177.37 ± 6.37	-274.52 ± 53.4
Adversary	-3.97 ± 3.71	$\mathbf{-2.43 \pm 5.88}$	-4.09 ± 2.35	-7.20 ± 4.60
Tag	2.10 ± 3.45	-0.90 ± 14.80	$\mathbf{4.08 \pm 2.65}$	1.15 ± 2.65

MAPPOQ exhibited the lowest average returns out of all the algorithms in three of the four MPE scenarios. It also displayed very high levels of variance across its five training seeds in both the Speaker Listener and Spread tasks, especially when compared to the baseline algorithms, whom each achieved an average return that was consistently close to their respective maximums.

Interestingly, MAPPOQ appears to have performed far better in the competitive environments of Adversary and Tag, a finding that is perhaps consistent with its more robust performance on the LBF tasks. It suggests that when the agents share the same rewards, MAPPOQ struggles to efficiently learn effective behaviours, requiring many more training samples before converging, given how far its average returns are from its maximum seen above.

Indeed, from the low averages, we infer that the sample efficiency of MAPPOQ is much lower than the other algorithms' in the cooperative environments. Furthermore, we speculate that the variance observed is linked to the centralised critics' value estimates, which we will discuss more in the following chapter.

5.2.3 Compute Time

Table 5.6: Average wall-clock time in seconds and 95% confidence interval reported over five training seeds for each algorithm on the MPE tasks. Times in bold represent the fastest time for that task.

Task	IA2C	MADDPG	MAPPO	MAPPOQ
Speaker Listener	$\mathbf{474.75 \pm 0.39}$	1634.39 ± 12.74	2537.44 ± 54.80	5307.20 ± 697.47
Spread	$\mathbf{707.42 \pm 2.51}$	3251.93 ± 483.82	5461.52 ± 800.86	10799.54 ± 697.28
Adversary	$\mathbf{644.83 \pm 4.47}$	2712.43 ± 50.88	4813.50 ± 84.58	5690.84 ± 161.34
Tag	$\mathbf{893.57 \pm 1.60}$	4759.37 ± 614.80	7396.06 ± 1145.70	9640.53 ± 613.00

As was found within the LBF tasks, the computational time increased with the number of agents in the environment. Additionally, we again observe that the CTDE algorithms are far slower than the fully decentralised IA2C, suggesting that including the additional information in the critics increases the cost of the training process.

Unlike the LBF tasks, the MPEs provides a distinct comparison between cooperation vs competition. The results support our speculation; compared to the other algorithms, MAPPOQ displays worse performance on the cooperative tasks than the competitive ones. For example, in Speaker Listener, MAPPOQ took more than double the time to complete training than the other algorithms.

Chapter 6

Discussion

6.1 Evaluation of Results

The goal of the project was to implement at least one stochastic RL algorithm with the MAAC method and compare it empirically against MADDPG on different discrete MARL environments. Motivating our goal was the suggestion from previous work [35] that Gumbel-Softmax, used by MADDPG to maintain its differentiability in discrete action spaces, causes a reparameterisation bias that worsens its performance in terms of both maximum and average returns. Therefore, comparing our alternate MAAC algorithm that does not require a Gumbel-Softmax to MADDPG would test their claims and discover the full performance implications of the reparametrisation bias.

Accordingly, within our project, we proposed MAPPOQ, an algorithm that employs a stochastic policy while also being a MAAC method. Furthermore, MAPPOQ utilised the PPO surrogate objective function, a feature that was included to account for the differences in sampling efficiencies between on-policy and off-policy algorithms (e.g. [35]). We then outlined several LBF and MPE tasks that would enable various comparisons to be drawn between MAPPOQ and the baseline algorithms across different environmental properties, such as sparsity of rewards, cooperation, competition, and observability.

Our results provided evidence to suggest that MADDPG does indeed suffer from a hindered performance under the metrics we measured within some of the discrete environments. However, and perhaps more interestingly, the results also pointed to a more nuanced trade-off for the MAAC method; despite its consequences for performance, Gumbel-Softmax allows deterministic policies to be used in discrete action spaces. Something that should not be understated because, as our results showed, stochastic

policies combined with the MAAC method often exhibit high variances. This result is also supported by the work of Lyu et al. [27], where the authors proved that CTDE suffers from higher variances than fully decentralised methods.

We hypothesise that the issues faced by MAPPOQ are general to all stochastic MAAC algorithms. Our justification for this comes down to the stochasticity of the action sampling. When each actor samples its following action, each action will include some associated variance given the probability distribution from which it was sampled. Therefore, combining these actions then combines the associated variances. This hypothesis is supported by results from other CTDE algorithms. For example, the COMA algorithm which has been shown to perform poorly on many multi-agent tasks [47, 35].

Moreover, given that in MAAC, the policy gradient estimate uses the critic's value judgement, we arrive at the issue for stochastic MAAC methods. The policies are estimated with a highly variable value from timestep to timestep, causing, as we saw in our results, highly variant policies that require many samples to begin to solve tasks. Furthermore, increasing the number of agents within the task would only increase the variance in the joint action.

Something we found true, especially in the Spread task where MAPPOQ achieved lower average returns than all baseline algorithms and displayed very high levels of variance across the five random seeds.

6.2 Project Limitations

Our research was limited in several ways; we discuss these limitations to provide a more accurate context around our findings. Additionally, we also believe that outlining the challenges faced by the project will help improve the current state of MARL research.

Firstly, our experiments focused on two MARL environments. Further insights could be found between our proposed algorithm and MADDPG by experimenting on a more diverse range of MARL tasks. For example, SMAC and multi-agent robot warehouse (RWARE) require more complex behaviour to be solved than both LBF and MPEs [36, 35].

Secondly, it is known that differences in the implementations of algorithms can cause a difference in observed performances. Therefore, our comparisons between the baselines may be limited because our implementations were not standardised. Lack of

standardised implementations for MARL algorithms is a known problem that causes differences in observed results [3], however work is being carried out to fix this issue and we hope that this gap can continue to be closed [35].

Thirdly, we performed a grid search over a pre-selected range of hyperparameters. Further investigation of hyperparameters could make use of either a random search over a wider range of hyperparameters or Bayesian optimisation, two methods shown to both perform more efficiently than grid search for optimal hyperparameter discovery [6, 41].

Lastly, our work compared an on-policy algorithm to the off-policy MADDPG. Off-policy algorithms are more sample efficient and often require more compute time than their on-policy counterparts. Thus, while utilising the PPO surrogate objective function within MAPPOQ did account for differences in sample efficiency, future investigations may want to compare an off-policy stochastic MAAC algorithm against MADDPG.

Chapter 7

Conclusions

In conclusion, we have proposed an alternative MAAC algorithm named MAPPOQ and compared it empirically against MADDPG on a diverse range of discrete-action multi-agent tasks. While MAPPOQ did not outperform MADDPG across all tasks, our findings did answer our research questions.

Our aim for the project was to investigate the hypothesis that Gumbel-Softmax hinders MADDPG's performance. Additionally, we hoped to obtain an algorithm that would outperform MADDPG or provide similar results with less required hyperparameter tuning. Our results showed that the use of a Gumble-Softmax does indeed hinder MADDPG on discrete-action spaces. However, perhaps more surprisingly, we discovered that MADDPG outperformed its stochastic alternative across many tasks, particularly those requiring more cooperation between agents. Thus, we speculate that a more nuanced trade-off exists between implementing MAAC with DDPG and implementing it with a stochastic policy gradient algorithm such as PPO.

We leave it to future work to investigate this trade-off further; where is the exact limit that DDPG becomes preferable over a stochastic algorithm implementation of MAAC, for example? Future work may also investigate methods for reducing the reparameterisation bias caused by Gumbel-Softmax or find ways to remove its use altogether while upholding deterministic action selection.

Bibliography

[1] Stefano V. Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *AAMAS '13 Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1155–1156. International Foundation for Autonomous Agents and Multi-Agent Systems, 2013.

[2] Stefano V. Albrecht and Peter Stone. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '17, page 547–555, Richland, SC, 2017. International Foundation for Autonomous Agents and Multi-Agent Systems.

[3] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021.

[4] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[5] Richard E Bellman et al. Dynamic programming, ser. *Cambridge Studies in Speech Science and Communication. Princeton University Press, Princeton*, 1957.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[7] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[8] Di Cao, Weihao Hu, Junbo Zhao, Qi Huang, Zhe Chen, and Frede Blaabjerg. A multi-agent deep reinforcement learning based voltage regulation using coordinated pv inverters. *IEEE Transactions on Power Systems*, 35(5):4120–4123, 2020. doi: 10.1109/TPWRS.2020.3000652.

[9] Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *34th Conference on Neural Information Processing Systems*, 2020.

[10] Filippos Christianos, Georgios Papoudakis, Arrasy Rahman, and Stefano V. Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *International Conference on Machine Learning (ICML)*, 2021.

[11] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pages 2177–2182. IEEE, 2012.

[12] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines, 2017.

[13] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[14] Jakob N Foerster. *Deep multi-agent reinforcement learning*. PhD thesis, University of Oxford, 2018.

[15] Jakob N Foerster, Yannis M Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, page 2145–2153, 2016.

[16] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[17] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

[18] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

[19] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[20] Wei Hu, Lechao Xiao, and Jeffrey Pennington. Provable benefit of orthogonal initialization in optimizing deep linear networks. In *In International Conference on Learning Representations*, 2020.

[21] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations*, 2017.

[22] Emilio Jorge, Mikael Kågebäck, Fredrik D Johansson, and Emil Gustavsson. Learning to play guess who? and inventing a grounded language as a consequence. *arXiv preprint arXiv:1611.03218*, 2016.

[23] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[24] T. Lillicrap, Jonathan J. Hunt, A. Pritzel, N. Heess, T. Erez, Yuval Tassa, D. Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

[25] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[26] Ryan Lowe, YI WU, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[27] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. *Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning*, page 844–852. International Foundation for Autonomous Agents and Multi-Agent Systems, 2021.

[28] Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

[29] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *Knowledge Engineering Review*, 27(1):1–31, 2012.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[31] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[32] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[33] Rémi Munos. Policy gradient in continuous time. *Journal of Machine Learning Research*, 7:771–791, 2006.

[34] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1906.04737*, 2019.

[35] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[36] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr,

Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[37] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[40] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[41] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[42] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[43] Richard S Sutton. Introduction: The challenge of reinforcement learning. In *Reinforcement Learning*, pages 1–3. Springer, 1992.

[44] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[45] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer, 1999.

[46] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[47] Bozhidar Vasilev, Tarun Gupta, Bei Peng, and Shimon Whiteson. Semi-on-policy training for sample efficient multi-agent policy gradients. *arXiv preprint arXiv:2104.13446*, 2021.

[48] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[49] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[50] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[51] David H Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pages 355–369. World Scientific, 2002.

[52] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of mappo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

[53] Yu Zhang, Zhiyu Mou, Feifei Gao, Jing Jiang, Ruijin Ding, and Zhu Han. Uav-enabled secure communications by multi-agent deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, pages 11599–11611, 2020.