Revisiting Discrete Gradient Estimation in MADDPG

Callum Tilbury

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2022

Abstract

MADDPG is an algorithm in Multi-Agent Reinforcement Learning that extends the popular single-agent method, DDPG, to multi-agent scenarios. Importantly, DDPG is an algorithm designed for continuous-action spaces, where the gradient of the state-action value function exists. For this algorithm to work in discrete-action spaces, then, discrete gradient *estimation* must be performed. For MADDPG, this is done using the Gumbel-Softmax estimator—a reparameterisation which relaxes a discrete distribution into a somewhat-similar continuous one. This method, however, is statistically biased, and some authors believe that this bias makes MADDPG perform poorly in grid-world situations, where the action-space is discrete. Fortunately, many alternatives to the Gumbel-Softmax exist, boasting a wide range of properties. This project details a handful of these alternatives, sourced from the literature, and integrates them into MADDPG for discrete grid-world scenarios. The corresponding impact on various performance metrics is then measured and analysed. It is found that one of the proposed estimators performs significantly better than the original Gumbel-Softmax in several tasks, both in the returns achieved and the time it takes for the training to converge. Based on these results, it is argued that a rich field of future work exists.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

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.

(*Callum Tilbury*)

Acknowledgements

And present gratitude
Insures the future's good,
And for the things I see
I trust the things to be;

— John Greenleaf Whittier

For a year such as this, I owe innumerable thanks. Some days I can hardly believe my fortune in such special friends, such an interesting course, and such a beautiful city. My silly words feel mostly futile, but ought to be shared nonetheless:

To the Skye Foundation—
Thank you for showering me with generosity and unlocking an unequivocally life-changing experience.

To Dr Stefano Albrecht—
Thank you for your guiding wisdom in this project.

To Filippos Christianos—
Thank you for your mentorship and advice, and for being so willing to share it.

To Claude and Arnu, from InstaDeep—
Thank you for the occasional consultation and excitement about this project. I can't wait for next year!

To my gorgeous friends in Edinburgh—
Thank you for being in community with me. Thank you for the meals, the laughs, and the love. I'll see you all in Cape Town!

Finally, to my ever-supportive cheerleaders back home:
my wonderful friends and my loving family—
Thank you for being on this journey with me, albeit from afar. I miss you all dearly.

Table of Contents

Acronyms & Initialisms

CTDE Centralised Training, Decentralised Execution

DDPG Deep Deterministic Policy Gradient
DPG Deterministic Policy Gradient
DPGT Deterministic Policy Gradient Theorem

GRMC Gumbel-Rao Monte Carlo
GS Gumbel-Softmax
GST Gapped Straight-Through

i.i.d. independent and identically distributed

LBF Level-Based Foraging

MADDPG Multi-Agent Deep Deterministic Policy Gradient
MARL Multi-Agent Reinforcement Learning
MPE Multi-Agent Particle Environment

POSG Partially-Observable Stochastic Game

RL Reinforcement Learning
RWARE Robot Warehouse

SPGT Stochastic Policy Gradient Theorem
STGS Straight-Through Gumbel-Softmax

TAGS Temperature-Annealed Gumbel-Softmax

Chapter 1

Introduction

1.1 Motivation

In recent years, interest in the field of Reinforcement Learning (RL) has grown markedly. Though in existence for many decades, the discipline's recent integration with deep learning—often called *deep* RL—has catalysed a renewed hope for its capabilities. Such excitement is certainly warranted: deep RL algorithms have been excelling consistently on a wide range of challenges, many of which seemed unthinkable in the past. Commonly cited feats include conquering many popular games, both modern and ancient [Vinyals et al., 2019, Berner et al., 2019, Wurman et al., 2022, Silver et al., 2016, Schrittwieser et al., 2020].

An important type of problem in RL is where not only a single agent acts, but *multiple* agents. These agents act together, either adversarially, co-operatively, or some combination thereof. Broadly, this paradigm is termed Multi-Agent Reinforcement Learning (MARL). Algorithms developed for single-agent contexts can be applied for multiple agents, where each agent simply acts independently, but performance here has been shown to be limited [Papoudakis et al., 2021]. Problems with such an approach include agents perceiving the environment as non-stationary [Papoudakis et al., 2019], and the so-called 'curse of dimensionality' [Du and Ding, 2021]. As an alternative, researchers have developed MARL-specific algorithms—either by extending extant single-agent approaches to multi-agent scenarios, or by developing new algorithms altogether.

One of the earliest approaches proposed for *deep* MARL—that is, MARL with the integration of deep learning—was the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm, by Lowe et al. [2017]. In this work, the authors extended

the single-agent Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2016] method, which is itself an extension of the Deterministic Policy Gradient (DPG) [Silver et al., 2014] method, to multi-agent scenarios. Crucially, DPG and its descendants are designed to work only with *continuous*-action spaces—where each action comes from an uncountable, continuous domain, e.g. the torque applied to motor. The alternative is a *discrete*-action space, which has countable set of possibilities, e.g. choosing to go up or down. The restriction to continuous domains is because the gradient of the 'state-action' value function (which indicates how valuable it is to execute a given action from a given state) taken with respect to the action, must exist. This is not the case for a discrete-action context.

Despite this, it seems that the authors of MADDPG desired a universal algorithm, one which could be applied to both continuous and discrete problems, while still building on the foundations of DPG. To enable MADDPG to work in discrete situations, then, a mathematical trick was applied: the Gumbel-Softmax (GS) reparameterisation. Essentially, this trick 'relaxes' the discrete, non-differentiable action-space into a somewhat-equivalent, continuous space—thus allowing an approximation of the gradient to exist. We term this a 'discrete gradient estimation' method. Relaxing the space in this way, however, introduces statistical *bias* into the gradient computation.

Recently, a benchmarking paper by Papoudakis et al. [2021] found that MADDPG achieved decent performance in certain MARL environments, but performed markedly worse in *grid-world* situations—where the action space is discrete. The authors suggested that this degradation of performance may be due to the aforementioned bias introduced by the GS.

Interestingly, the GS—and the field of discrete gradient estimation, more broadly—appears in a host of contexts outside of MARL. As a result, a wealth of alternatives has been proposed for the GS, many of which focus on lowering the bias it introduces. As of yet, though, it seems that not many of these techniques have been integrated into MARL, and certainly not into MADDPG. Doing so, however, may be a fruitful endeavour, and would provide a way to explore the conjecture made by Papoudakis et al. [2021]. This then motivates our current investigation.

1.2 Goals

With this context in mind, we now encapsulate our enquiry in a question:

> *Can alternative discrete gradient estimation methods improve the performance of MADDPG in discrete-action space environments, when compared to the original Gumbel-Softmax reparameterisation?*

In answering this question, we hope to make three broad contributions:

- A synthesis of the literature, focusing on a set of available discrete gradient estimation methods which *could* be integrated into MADDPG.
- An implementation of a handful of such methods into MADDPG, with an investigation into whether improvements occur when these methods are applied in discrete-action spaces.
- If successful, a cursory analysis into *why* such improvements occur.

1.3 Structure

This report proceeds as follows. Chapter 2 provides the necessary foundations of the project, broadly covering the important background information—here, we discuss the topics of gradient estimation and RL, and then we synthesise them: looking at how the former appears in the latter. Chapter 3 dives deeply into *discrete* gradient estimation, looking firstly at the GS method, and then the proposed alternatives from the literature. A few proposals are selected for further investigation, and the theory behind these approaches is presented in more detail. Chapter 4 proceeds with the experimental methodology for the project, detailing the environments used for evaluation, and how 'performance' is defined. This is followed by the results, presented in Chapter 5, where discussion is also provided. Finally, in Chapter 6, conclusions are drawn, and recommendations for future work are made.

Chapter 2

Foundations

This chapter aims to provide the necessary groundwork upon which the rest of the project builds. We begin with an overview of the challenge of gradient estimation for an expected cost—a key theoretical backbone for our work. We then provide a short introduction to some RL concepts, and define our notation. Thereafter, we narrow our discussion to a relevant class of RL algorithms, under which MADDPG falls, and show where gradient estimation is relevant. Finally, we present the core MADDPG algorithm, highlighting the focus of this investigation.

2.1 Primer on Gradient Estimation

Methods in machine learning often require the calculation of a gradient, for the sake of gradient-based optimisation [Goodfellow et al., 2016, chpt. 6]. Note that this field is far richer than the brief treatment given here; for a comprehensive discussion, see the review by Mohamed et al. [2020]. For now, consider a random vector, x, which is drawn from some parametric distribution, $x \sim p(x;\theta)$, where θ represents the parameters. Suppose there is some cost function, $f(x)$, and we wish to minimise the expected cost with respect to θ, via gradient descent. To do this, we require an estimate of $\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)]$. Expanding the expected value into its integral form:

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] = \nabla_\theta \int_x p(x;\theta) f(x)\, \mathrm{d}x = \int_x [\nabla_\theta p(x;\theta)] f(x)\, \mathrm{d}x \tag{2.1}$$

In most cases, such an integral is intractable in closed-form. Furthermore, Monte Carlo methods cannot be used to estimate its value, since we cannot sample from $\nabla_\theta p(x;\theta)$, as it is not necessarily a valid probability distribution—it could, for example, be negative for some value of x.

The literature primarily suggests two classes of techniques to overcome this problem, both of which rearrange the integral into a form with which Monte Carlo estimation can be performed.

Score Function Estimation

The first approach is called *Score Function Estimation* [Kleijnen and Rubinstein, 1996], also known as the *Likelihood Ratio* [Glynn, 1990] approach, or just *REINFORCE* [Williams, 1992]. This technique hinges on the following relationship:

$$\nabla_\theta p(x;\theta) = p(x;\theta) \cdot \frac{\nabla_\theta p(x;\theta)}{p(x;\theta)} = p(x;\theta)\nabla_\theta \log p(x;\theta) \tag{2.2}$$

Substituting this equivalence into the integrand from (2.1), we obtain:

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] = \int_x p(x;\theta)\nabla_\theta \log p(x;\theta) f(x)\, \mathrm{d}x \tag{2.3}$$

Because $p(x;\theta)$ is, by definition, a valid probability distribution, we can change this integral back into an expectation under $p(x;\theta)$,

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] = \mathbb{E}_{x\sim p(x;\theta)}\Big[\nabla_\theta \log p(x;\theta) f(x)\Big] \tag{2.4}$$

and thus use a Monte Carlo estimate with N samples,

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \log p(x^{(n)};\theta) f(x^{(n)}) \quad , \quad x^{(n)} \sim p(x;\theta) \tag{2.5}$$

Notice that for the estimate in (2.5) to exist, the gradient of the log of the distribution must exist, but no constraints are placed on $f(x)$. This is beneficial, for $f(x)$ may be a 'black box'—we can simply query f for each sampled value of x. However, this method is known to have high variance, particularly when implemented in high-dimensional problems [Mohamed et al., 2020].

Pathwise Derivative Estimation

The second approach to overcome the challenges encountered in (2.1) is called *Pathwise Derivative Estimation*. Fundamental to this method is the 'reparameterisation trick' [Kingma and Welling, 2013]. Recall that we previously defined a θ-parameterised *distribution*, $p(x;\theta)$. Now, we decouple the parameterisation from the randomness, by first defining a random variable ε, and then defining a θ-parameterised *transformation*, t. That is, to sample x:

$$x = t(\varepsilon;\theta) \quad , \quad \varepsilon \sim p(\varepsilon) \tag{2.6}$$

The so-called 'Law of the Unconscious Statistician' [DeGroot and Schervish, 2011, pg. 213] implies that the expected value of the cost under the original parameterisation is the same as that under the reparameterisation. That is,

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] \overset{\text{LOTUS}}{=} \nabla_\theta \mathbb{E}_{\varepsilon\sim p(\varepsilon)}[f(t(\varepsilon;\theta))] \tag{2.7}$$

Accordingly, we can expand the expectation under the new parameterisation into its integral form, and then apply the chain rule:

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] = \nabla_\theta \int_\varepsilon p(\varepsilon) f(t(\varepsilon;\theta))\, \mathrm{d}\varepsilon \tag{2.8}$$

$$= \int_\varepsilon p(\varepsilon) \nabla_\theta f(t(\varepsilon;\theta))\, \mathrm{d}\varepsilon \tag{2.9}$$

$$= \int_\varepsilon p(\varepsilon) \nabla_\theta t(\varepsilon;\theta) \nabla_x f(x)\big|_{x=t(\varepsilon;\theta)}\, \mathrm{d}\varepsilon \tag{2.10}$$

As before, $p(\varepsilon)$ is a valid probability distribution by definition, and we can thus rewrite the expression as an expectation, and take a Monte Carlo estimate with N samples:

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] = \mathbb{E}_{\varepsilon\sim p(\varepsilon)}\left[\nabla_\theta t(\varepsilon;\theta) \nabla_x f(x)\big|_{x=t(\varepsilon;\theta)}\right] \tag{2.11}$$

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta t(\varepsilon^{(n)};\theta) \nabla_x f(x^{(n)})\big|_{x^{(n)}=t(\varepsilon^{(n)};\theta)} \quad , \quad \varepsilon^{(n)} \sim p(\varepsilon) \tag{2.12}$$

This method for estimating the gradient is known to have lower variance than the score-function approach, even for high-dimensional problems [Mohamed et al., 2020]. However, notice the key difference between the estimate from before, in (2.5), and this estimate, in (2.12). Whereas no constraints were placed on $f(x)$ previously, this function must now be differentiable with respect to x. That is, $\nabla_x f(x)$ must exist, and as a result, $f(x)$ can no longer simply be a 'black box'. Alas, in many contexts, this is problematic. Relevant to this work is the situation where $f(x)$ is not differentiable due to discreteness—i.e. where the cost function is defined as $f : \mathbb{Z}^n \mapsto \mathbb{R}$.

Introduced concurrently by Jang et al. [2017] and Maddison et al. [2017], the Gumbel-Softmax (GS) (or 'Concrete') distribution is an attempt to solve this issue of non-differentiability. This approach 'relaxes' a discrete, non-differentiable distribution into a continuous, differentiable distribution, which is approximately equivalent. In doing so, a reparameterisation gradient can be admitted, for Pathwise Derivative Estimation. Given the importance of the GS to this project, robust mathematical details will be given in Chapter 3.

2.2 Reinforcement Learning Background

Solving reinforcement learning problems, whether the multi-agent case presented in this dissertation, or the single-agent case, has been a topic of research for many decades (e.g. [Waltz and Fu, 1965]). Naturally, from these efforts, a myriad of techniques and algorithms has arisen. For a comprehensive treatment of the discipline, starting from the foundations and building in complexity, the reader is encouraged to see the canonical textbook by Sutton and Barto [2018]. Now, we focus our attention on a subset of relevant methods, and do so briefly.

First, we ought to formalise the discussion, adopting common notation [Christianos et al., 2020, Lowe et al., 2017]. We model a multi-agent problem as a Partially-Observable Stochastic Game (POSG) [Shapley, 1953, Hansen et al., 2004], operating in discrete time-steps, with a set of N agents, $\mathcal{N} = \{1, \dots, N\}$. Let the state space be denoted as $\mathcal{S}$, the joint-action space as $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$, and the joint-observation space as $O = O_1 \times \cdots \times O_N$. Each agent $i \in \mathcal{N}$ perceives only a local observation, $o_i \in O_i$, which depends on the current state and the joint-action taken—we denote this as the agent's observation function, $\Omega_i : \mathcal{S} \times \mathcal{A} \mapsto \Delta(O_i)$, where Δ indicates the probability simplex of the appropriate dimension, i.e. $\Delta^{\dim(O_i)-1}$. We define a transition function, $P : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$, which describes the probability of transitioning from one state to another, given a joint action. We further define a reward function for each agent, $\mathcal{R}_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$. Let the reward given to agent i at time-step t be denoted as $r_i^{(t)}$. The game begins in an initial state, which depends on the distribution $\rho = \Delta(\mathcal{S})$.

We consider here 'model-free' approaches, where agents do not explicitly create nor store a model of the environment's underlying dynamics. Instead, the agents repeatedly interact with the environment and try learn an optimal way of doing so. The manner in which an agent acts is termed its 'policy'—what it should do when it is in a particular state. Informally speaking, agents are trying to figure out a 'good' policy—i.e. one which yields high rewards. We define the 'return' for an agent i as its discounted cumulative reward, $G_i = \sum_{t=0}^{T} \gamma^t r_i^{(t)}$, where T is the number of time-steps in an episode, and $\gamma \in (0, 1]$ is a discounting factor—controlling how much we care about future rewards relative to current rewards.

Denote each agent's policy as π_i, with the set of all policies being $\pi = \{\pi_1, \dots, \pi_N\}$. The objective in MARL, then, is to find policies such that the return of each agent i, following π_i, is maximised with respect to the other agents' policies, $\pi_{-i} := \{\pi \setminus \pi_i\}$.

That is, we aim to find an optimal set of policies, π, such that

$$\forall i \ : \ \pi_i \in \arg\max_{\hat{\pi}_i} \mathbb{E}\left[G_i \mid \hat{\pi}_i, \pi_{-i}\right] \tag{2.13}$$

How can we learn such an optimal policy? Broadly, two model-free philosophies have dominated the field: *Value Function* methods, and *Policy Gradient* methods.

Value function methods involve, perhaps obviously, 'values': metrics that indicate the rewards an agent may expect to receive when in a given state, taking a given action, or both. We often care about the *state-action value*—also known as the Q-value. For a set of policies π, the Q-value for agent i, taking an action a_i, from a state s_i, is defined as:

$$Q^{\pi}(s_i, a_i) := \mathbb{E}_{\pi}[G_i \mid s_i, a_i] \tag{2.14}$$

In value function methods, an agent learns these values for various states and actions, and thereafter generates a policy *implicitly*—e.g. the common 'epsilon greedy' approach selects a *random* action occasionally (with a probability of ε), and the 'greedy' action (i.e. the action with the highest value) otherwise. A popular instance of the value-function technique is Q-learning [Watkins, 1989, Watkins and Dayan, 1992].

Early methods, Q-learning included, attempted to learn in a *tabular* manner; that is, by enumerating the state and action spaces, and explicitly storing the learned values for every state-action pair. Such methods, however, do not scale well: problems quickly become infeasible as the problem's dimensionality increases. Moreover, experience is not used efficiently, for no interpolation of values of nearby states can be made, despite this being a sensible technique [Kaelbling et al., 1996]. An alternative is to use a function approximator, with parameters θ. For example, rather than storing many *Q-values*, we could learn a *Q-function*, $Q(s, a; \theta)$.

In a landmark paper by Sutton et al. [1999], the authors took the function approximation route, but with a key difference: the policy is modelled *explicitly*. That is, instead of learning values and using them to generate a policy, an agent's policy is encoded directly as $\pi_i(a \mid s; \theta)$—a distribution over actions given the state. Assuming the policy is differentiable with respect to its parameters (i.e. $\frac{\partial \pi(a|s;\theta)}{\partial \theta}$ exists), an optimal policy can be found through gradient ascent of the expected return. This work ushered in the other dominant approach to model-free learning: policy gradient methods.

Modern reinforcement learning algorithms often use a combination of value- and policy-based approaches, broadly termed *Actor-Critic* methods—whereby the 'actor' learns the policy function, and the 'critic' learns the value function (e.g. A3C [Mnih et al., 2016]).

2.3 Deterministic Policy Gradient Methods

We now narrow our discussion to a specific class of RL algorithms, called *deterministic* policy gradient methods, and synthesise this topic with the gradient estimation theory from Section 2.1.

By modelling the policy with a function approximator, as $\pi(a \mid s;\theta)$, the goal is to use gradient-based optimisation to maximise the expected returns under this policy with respect to the policy parameters, θ. A fundamental result to enable such methods is the Stochastic Policy Gradient Theorem (SPGT), presented by Sutton et al. [1999]:

$$\nabla_\theta \mathbb{E}_\pi[r(s,a)] = \mathbb{E}_\pi[\nabla_\theta \log \pi(a \mid s;\theta)\, Q^\pi(s,a)] \tag{2.15}$$

Notice the similarity of this equivalence, (2.15), and the score-function estimator presented in (2.4). Indeed, the former is an instance of the latter—the work by Sutton et al. [1999] builds on the REINFORCE method [Williams, 1992]. In fact, a simple way of implementing (2.15) is to approximate $Q^\pi(s,a)$ with the sample return, and this method itself is called REINFORCE in RL literature. As discussed previously, though such methods are relatively straightforward, they suffer from high variance.

Taking a different approach, Silver et al. [2014] introduced the Deterministic Policy Gradient (DPG) method. Here, instead of trying to learn a *stochastic* policy, $\pi(a \mid s;\theta)$, the authors make the policy *deterministic*, notating it as $a = \mu(s;\theta)$. In doing so, they derive the Deterministic Policy Gradient Theorem (DPGT):

$$\nabla_\theta \mathbb{E}_\mu\big[r(s,a=\mu(s;\theta))\big] = \mathbb{E}_\mu\Big[\nabla_\theta \mu(s;\theta)\, \nabla_a Q^\mu(s,a)\Big|_{a=\mu(s;\theta)}\Big] \tag{2.16}$$

We notice again a similarity, now between (2.16) and the pathwise-derivative method in (2.11). Silver et al. [2014] show that, indeed, the DPGT is the limiting case of a *reparameterisation* of the SPGT. The benefit of DPG is thus the same as for pathwise-derivative methods generally—reduced variance in the gradient estimates [Mohamed et al., 2020]. However, the same drawback exists too; notice how the gradient $\nabla_a Q^\mu(s,a)$ must now exist. Herein lies a limitation of DPG: one cannot use the method in discrete-action problems, for this gradient does not exist in such contexts.

The authors of DPG made no mention of this topic, for their method was presented explicitly for continuous-action problems, where the required gradients do exist. Building on the DPG algorithm—inspired by success of Mnih et al. [2013] in incorporating deep neural networks to Q-learning—Lillicrap et al. [2016] then developed the Deep Deterministic Policy Gradient (DDPG) algorithm. This approach introduced, amongst

other things, the use of deep neural networks for the function approximation in DPG. Here, too, the focus was explicitly on continuous action-spaces, and there were no problems with the aforementioned gradient calculations.

The MADDPG algorithm [Lowe et al., 2017] was introduced shortly thereafter, as a multi-agent application of DDPG. Unlike DPG and DDPG, though, it seems that the authors desired a unified algorithm: something that could be used in both continuous and discrete situations. For continuous contexts, the underlying DPG approach could work as per usual; for discrete contexts, however, the authors applied the GS trick [Jang et al., 2017, Maddison et al., 2017] to the discrete actions taken. In doing so, the discrete action distribution was relaxed, and an *approximation* of the gradient, $\nabla_a Q^\mu(s,a)$, could be used.

Though this relaxation 'works'—i.e. it enabled the authors to train MADDPG in discrete-action spaces—the GS is known to introduce statistical *bias* into the gradient estimation [Lorberbom et al., 2019]. We recall that, in their surveillance of MARL techniques, Papoudakis et al. [2021] found that MADDPG achieved competitive returns in some tasks, but performed poorly in grid-world environments, where the action-space is discrete. The authors believed that the poor performance was a consequence of the bias introduced by the GS.

2.4 MADDPG Implementation Details

Justification for using MADDPG

We choose to study the impact of using the GS in discrete-action spaces with *MADDPG*, rather than, e.g., with DPG or DDPG, for two reasons. Firstly, since the problem was identified in MADDPG by Papoudakis et al. [2021], we have a baseline from which to work—this includes a code implementation for the algorithm, from which inspiration can be drawn; already-tuned hyperparameters for a variety of tasks; and existing results with the GS, for comparison and sanity-checking. Secondly, MARL is an active field of research, and improving results in multi-agent tasks is a worthwhile endeavour. Therefore, if an improvement to the original MADDPG algorithm could be found—e.g. finding an alternative to the GS that yields better returns—it would be of great use to the broader MARL community.

Core Algorithm

Some high-level knowledge about MADDPG is helpful for our discussion. Introduced by Lowe et al. [2017], MADDPG pioneered as an early *deep* RL method for multi-agent settings. Situations with multiple agents are challenging because of environmental non-stationarity [Papoudakis et al., 2019]—that is, from the perspective of a single agent, the environment changes in a way that cannot be explained by that agent's own actions. Stationarity, however, is fundamental to many extant single-agent RL algorithms.

MADDPG overcomes this difficulty by using a Centralised Training, Decentralised Execution (CTDE) approach, where each agent has a *centralised* critic but a *decentralised* actor. By centralising the critic—i.e. by conditioning the training of the agents on the joint observations and actions—the problem can be perceived as stationary from the perspective of each agent. Each agent's actor network, however, is still only conditioned on that agent's local observation, such that at execution-time, the agents are indeed acting individually. For agent i, we write this as:

$$\text{Critic: } Q_i(o_1, \ldots, o_i, \ldots, o_N, a_1, \ldots, a_i, \ldots, a_N) \quad ; \quad \text{Actor: } \mu_i(o_i)$$

Since this deals with the problem of non-stationarity [Papoudakis et al., 2019], the single-agent RL technique of DDPG can be utilised in multi-agent scenarios—hence, MADDPG. We recall, however, that for *discrete*-action tasks used here, a relaxation of the action distribution must be applied, as discussed previously.

Other implementation details of MADDPG build on those for DDPG [Lillicrap et al., 2016]. These include: having separate 'behaviour' and 'target' networks with Polyak averaging to stabilise training, introduced by Mnih et al. [2013] and discussed analytically by Zhang et al. [2021]; and training with uniform samples from an experience replay buffer, $\mathcal{D}$, such that the correlations between consecutive training-steps are broken [Mnih et al., 2013].

The implemented algorithm for this project is kept mostly the same as it was in the original paper by Lowe et al. [2017], with some extensions and omissions, mentioned shortly. High-level details for our core implementation are given in Algorithm 1. Typeset in orange is the crux of this project: the gradient estimation required for the sake of updating the actor, since the agents are operating in discrete-action spaces. We notate here the estimation technique used as $\xi(\cdot)$. In the original implementation, this is simply the GS; this project thus hopes to find an improved $\xi(\cdot)$.

Algorithm 1: MADDPG algorithm for N agents in discrete-action spaces

Denote actor networks as: $\mu_i = \mu(\cdot\,;\theta_i)$, $\bar{\mu}_i = \mu(\cdot\,;\bar{\theta}_i)$, and
denote critic networks as: $Q_i = Q(\cdot\,;\phi_i)$, $\bar{Q}_i = Q(\cdot\,;\bar{\phi}_i)$,
where $\{\theta, \phi\}$ and $\{\bar{\theta}, \bar{\phi}\}$ indicate the *behaviour* and *target* parameters respectively.

while elapsed-time-steps $<$ total-time-steps **do**
  Initial observations: $\boldsymbol{o} = \{o_1, \ldots, o_N\}$
  for $t = 1$ to T **do**
    Execute actions with each agent's policy: $\boldsymbol{a} = \{\mu_1(o_1), \ldots, \mu_N(o_N)\}$
    Receive rewards $\boldsymbol{r}$ and new observations $\boldsymbol{o'}$
    Store tuple $\{\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{o'}\}$ in replay buffer $\mathcal{D}$
    Update current observations: $\boldsymbol{o} \leftarrow \boldsymbol{o'}$

    Sample a random mini-batch of S samples $\{\boldsymbol{o}^j, \boldsymbol{a}^j, \boldsymbol{r}^j, \boldsymbol{o'}^j\}$ from $\mathcal{D}$
    for agent $i = 1$ to N **do**
      Set target using current reward and target networks:

$$y_i^j = r_i^j + \gamma \bar{Q}_i\left(\boldsymbol{o'}^j, \bar{\mu}_1(o_1'^j), \ldots, \bar{\mu}_N(o_N'^j)\right)$$

      Update critic by minimising the loss, $\mathcal{L}_c$:

$$\nabla_{\phi_i} \mathcal{L}_c(\phi_i) = \nabla_{\phi_i} \frac{1}{S} \sum_j \left(y_i^j - Q_i(\boldsymbol{o}^j, \boldsymbol{a}^j)\right)^2$$

      Update actor using the sampled policy gradient:

$$-\nabla_{\theta_i} \mathcal{L}_a(\theta_i) = \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j)\, \nabla_{a_i} Q_i(\boldsymbol{o}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \xi\left(\mu_i(o_i^j)\right)}$$

    end for
    Update target parameters for each agent i using soft updates of size β:

$$\bar{\theta}_i \leftarrow \beta\theta_i + (1-\beta)\bar{\theta}_i$$
$$\bar{\phi}_i \leftarrow \beta\phi_i + (1-\beta)\bar{\phi}_i$$

  end for
end while

Extensions & Omissions

In the period since MADDPG was originally introduced, extensions to the algorithm have been proposed by various authors. The extensions implemented for this project mostly follow the lead of Papoudakis et al. [2021] in their benchmarking paper, and the salient ones are mentioned below:

- *Replay buffer warm-up*
  Early in the training, the replay buffer $\mathcal{D}$ contains only a small set of samples; to improve exploration early-on, the buffer is first populated with random transitions from untrained networks (e.g. [Srivastava et al., 2019]). After this, training is undertaken normally.

- *Policy regularisation*
  Adding a regularising term to each agent's policy has been shown to have some benefits, including promoting co-operation between them [Roy et al., 2020, Liu et al., 2021]. Specifically, with a regularisation parameter λ, we add a term to the actor loss such that the gradient is taken as: $-\nabla_{\theta_i}\left[\mathcal{L}_a(\theta_i)+\lambda\sum_j(\mu_i(o_i^j))^2\right]$.

- *Reward standardisation*
  In some environments, the received rewards can vary largely in magnitude, making learning unstable. This issue can be alleviated by normalising each reward by the running statistics of the rewards over each agent's lifetime [Papoudakis et al., 2021].

- *Increase training frequency*
  In the implementation by Lowe et al. [2017], the network parameters were updated after every 100 samples added to the replay buffer. Here, we increase that update frequency—the specifics of which depend on the environment.

Moreover, some aspects of MADDPG were *not* implemented here. Examples include policy ensembles [Lowe et al., 2017], parameter sharing (e.g. [Gupta et al., 2017, Chu and Ye, 2017]), and using Gated Recurrent Units [Cho et al., 2014] in the neural networks (e.g. [Papoudakis et al., 2021]). These omissions were made to avoid unnecessary complexity, for the present research enquiry is about the performance of the GS relative to other gradient estimation techniques, *not* about achieving maximal absolute returns. Advances in the current project could certainly be applied alongside other extensions for a wholly-stronger algorithm—this is left as future work.

Chapter 3

Discrete Gradient Estimation

We now provide an accessible explanation of the discrete gradient estimation techniques proposed for this project, starting with an exposition of the original GS method. A brief literature surveillance of alternative techniques is then given, followed by a description of the methods chosen for experimental investigation.

3.1 The Gumbel-Softmax

Fundamental to this topic, there are two questions: how does one sample from a discrete distribution, and how does one calculate the gradient of taking this sample? A common answer to the first question is an important precursor to the variety of answers for the second.

We consider a situation of a parametric discrete distribution, $p(a;\zeta)$, specified by an *unconstrained* vector of parameters, $\zeta \in \mathbb{R}^N$. In our context, these parameters represent the outputs of a policy network, and we wish to choose one of N possible actions based on their values—that is, sample $a \sim p(a;\zeta)$.

A simple method of sampling from this distribution is first to apply the softmax operation,

$$\omega_i = \text{softmax}(\zeta_i) := \frac{\exp \zeta_i}{\sum_{n=1}^{N} \exp \zeta_n} \tag{3.1}$$

to yield *constrained* parameters, ω: $\omega_i \geq 0$, $\sum_n^N \omega_n = 1$, which indicate the likelihood of each action. One can then divide the unit interval into 'chunks' proportional to the constrained values, and sample uniformly over this interval; the discrete sample is whichever chunk the uniform sample falls into [Barber, 2012, Chpt. 27].

An alternative is the so-called 'Gumbel-Max' trick (discussed by Maddison et al.

[2014]), which avoids computing the constrained probabilities. Instead, one simply perturbs each of the unconstrained parameters by noise drawn from a *Gumbel* distribution [Gumbel, 1954]. The discrete sample is whichever perturbed value is the largest:

$$a = \text{one\_hot}_N(\arg\max_i(\zeta_i + g_i)) \quad , \quad g_i \sim G(0,1) \tag{3.2}$$

where one_hot_N encodes the integer representation of the arg max as an N-dimensional vector, and $G(0,1)$ is the 'standard' Gumbel distribution. Drawing Gumbel noise can be done via inverse transform sampling: by first sampling from the uniform distribution, $u_i \sim U(0,1)$, and then transforming according to $g_i = -\log(-\log(u_i))$ [Murphy, 2023, Chpt. 6.5.6].

We now answer the second question from earlier: after taking a discrete sample, how do we calculate the gradient? The arg max's derivative is zero everywhere, except at the discontinuous transitions between discrete values, where the derivative is undefined. As a result, the Gumbel-Max trick is non-differentiable. We may wonder, can we 'smoothen' the arg max into something that *is* differentiable? Indeed, this was the key idea conceived by Jang et al. [2017] and Maddison et al. [2017] in presenting the Gumbel-Softmax (GS). As its name suggests, instead of using an arg max operation, a softmax is used—a *softer* arg max, in a sense. Furthermore, the softmax is *tempered* with a temperature parameter, $\tau > 0$: $\text{softmax}_\tau(x) = \text{softmax}(\frac{x}{\tau})$. In the limit of $\tau \to 0$, this operation is equivalent to the arg max, and thus the GS approaches the original distribution. Conversely, as $\tau \to \infty$, the GS approaches a uniform distribution, where each category is equally-likely. The temperature thus controls the 'degree of relaxation'.

Reusing the gradient-estimator notation $\xi(\cdot)$ from before, the relaxed distribution is:

$$\xi_{\text{GS}}(p(a;\zeta)) = \text{softmax}_\tau(\zeta_i + g_i) \quad , \quad g_i \sim G(0,1) \tag{3.3}$$

$$= \frac{\exp((\zeta_i + g_i)/\tau)}{\sum_{n=1}^{N} \exp((\zeta_n + g_n)/\tau)} \tag{3.4}$$

For later discussions, it is helpful to consider the GS as a relaxation over the N-dimensional probability simplex, Δ^{N-1}. For visualisation, consider the case of $N = 3$. Whereas previously we had a point-mass on the simplex for our parametric distribution, specified by ζ, we now have a smoothed probability *density*. This is shown in Figure 3.1, where the likelihood of the density corresponds to the degree of darkness, and the three sections indicate the three possible realisations of the discrete sample.

By relaxing the distribution in this way, it becomes differentiable—meaning we can incorporate it into a gradient-based optimisation procedure. There is a downside,

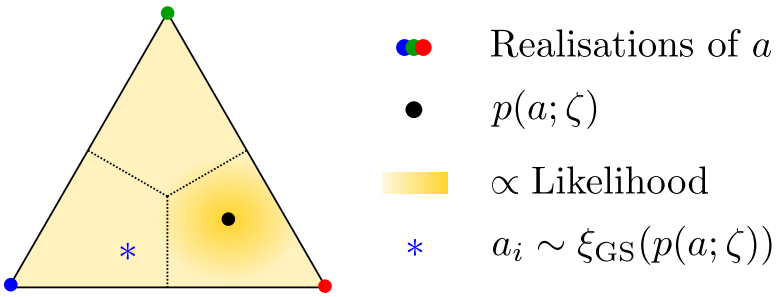


Figure 3.1: Depiction of the GS distribution over the probability simplex. An illustrative sample from the blue category is shown. (Adapted from Fan et al. [2022])

however: in relaxing, we introduce statistical bias [Paulus et al., 2021]. To understand this intuitively, consider again the limit $\tau \to \infty$, where the distribution becomes uniform. In such a case, we have removed all parametric information, ζ, about our problem—each category simply has a probability of $1/N$. Hence, as we relax, we also steer further away from the original distribution. Herein lies a trade-off: turning the temperature too low means having extreme gradients (or non-existent gradients when $\tau = 0$), but turning the temperature too high means introducing a large bias.

Though such a bias is inevitable when relaxing the distribution, there is an easy improvement to the vanilla GS. By naïvely applying the relaxation, we introduce bias in both the 'forward pass' (when we sample from the distribution) *and* in the 'backward pass' (when we calculate the gradients, e.g. for updating our neural network). However, it is only the latter that requires differentiability. Hence, building on the so-called 'Straight-Through' estimator proposed by Bengio et al. [2013], Jang et al. [2017] also introduce the Straight-Through Gumbel-Softmax (STGS) estimator—where, in the backward pass, the GS relaxation is applied, but in the forward pass, the original $\arg\max$ operation is used. In the context of this project, unless otherwise stated, we use GS to mean the STGS—e.g. in MADDPG, it is actually the straight-through variant, the STGS, that is applied, but we have hitherto called it the GS. Alas, despite presenting an unmodified forward computation, the STGS still introduces bias into the gradient estimation, with the bias dependent on the temperature used.

3.2 Available Alternatives

A core insight of this project is that discrete gradient estimation does not exist solely in the domain of RL. In fact, the original GS papers [Jang et al., 2017, Maddison et al., 2017] demonstrated the technique on problems such as structured output prediction

and density estimation. Indeed, discrete gradients arise in a wide variety of contexts—including discrete variational auto-encoders [Rolfe, 2017, Kingma et al., 2019], hard attention [Gulcehre et al., 2017, Yan et al., 2018], generative adversarial networks for text [Kusner and Hernández-Lobato, 2016, Zhang et al., 2017], and convolutional networks [Veit and Belongie, 2019]. As a result, the biased reparameterisation of the GS is problematic in a wide variety of domains, and accordingly, a significant research effort has focused on improving the method. In this brief section, we present a handful of salient papers that *could*, conceivably, be integrated into MADDPG, with a hope of overcoming the problems of the GS. For a more thorough analysis of the GS and associated advances in discrete gradient estimation, see the recent review by Huijben et al. [2022].

As elucidated in Section 2.1, there are two dominant approaches to gradient estimation generally [Mohamed et al., 2020]: score-function methods, and pathwise-derivative methods. Many algorithms in the literature avoid any discrete-gradient problems by sticking to the first approach, such that the SPGT from (2.15) can be used. Improvements in these methods focus on reducing the variance of the estimator, usually through the use of control variates. Examples include: subtracting a 'baseline' [Weaver and Tao, 2001, Greensmith et al., 2001]; using a Taylor expansion of a mean-field network, as in MuProp [Gu et al., 2016]; and using copula-based sampling, as in CARMS [Dimitriev and Zhou, 2021]. Authors have also *combined* score-function and pathwise-derivative methods, leveraging desirable qualities from both approaches. For example: using both REINFORCE and the GS in conjunction, as in REBAR [Tucker et al., 2017]; training a surrogate neural network as a control variate, as in RELAX [Grathwohl et al., 2018]; and using sampling without replacement [Kool et al., 2020].

Though these methods are promising, we choose here to focus on extensions that invoke the pathwise-derivative approach exclusively, such that we can give them a proper, thorough treatment. Early pathwise-derivative techniques focused on Bernoulli variables, e.g. the work by Bengio et al. [2013], with modern developments such as FouST [Pervez et al., 2020]. However, the GS trick [Jang et al., 2017, Maddison et al., 2017] was the first to formulate a pathwise-derivative gradient estimator for *categorical* variables. Since then, various improvements have been suggested to this end.

The Gumbel-Rao Monte Carlo (GRMC) approach [Paulus et al., 2021] applies 'Rao-Blackwellisation' to the original GS estimator, by drawing multiple Gumbel samples, and then marginalising them out. By the Rao-Blackwell Theorem [Blackwell, 1947], this estimator is usually better, and never worse, than the original estimator.

The Gapped Straight-Through (GST) estimator [Fan et al., 2022] builds on this work by using 'deterministic perturbations' instead of Gumbel noise. Here, estimator improvements are shown both analytically and empirically. The 'Invertible Gaussian Reparameterisation' [Potapczynski et al., 2020] is a slightly different approach, where *Gaussian* noise is used instead of Gumbel noise. Andriyash et al. [2019], too, move away from using Gumbel noise in their method, and propose a simple piecewise-linear relaxation instead. Some bolder methods exist too, which steer further away from the GS foundations. For example, Lee et al. [2018] generalise the reparameterisation trick, as discussed in (2.6), through manifold sampling, and are able to create an unbiased *and* reduced-variance estimator. Lorberbom et al. [2019] avoid the need to 'relax' the categorical distribution altogether by applying the technique of 'direct optimisation'.

Though many possible avenues of exploration exist, we focus on *two* of these novel options—these being, the GRMC and GST algorithms. We choose these in particular for they require straightforward changes, as they were developed explicitly as descendants of the GS. Furthermore, we also attempt to find simple adjustments to the extant GS method, such as using lower temperatures and temperature annealing, to see if such easy changes could improve the performance with MADDPG. The implemented methods are discussed in the following section.

3.3 Chosen Alternatives

Straight-Through Gumbel Softmax (STGS-1, STGS-T)

Interestingly, both in the original MADDPG paper [Lowe et al., 2017], as well as the benchmarking paper [Papoudakis et al., 2021], it seems that the authors simply use a temperature of 1.0 for the GS relaxation*; yet, Papoudakis et al. [2021] suggest the bias is the problem with the GS, which is related to the temperature. Accordingly, it seems worthwhile to explore alternative temperatures.

Under this heading, then, we consider two estimators. The first is the 'baseline' implementation, for it was implemented in past works: the STGS estimator with a temperature of $\tau = 1.0$, denoted as **STGS-1**. We further consider the STGS estimator with a temperature of $\tau < 1.0$, denoted as **STGS-T**, where τ is a tunable hyperparameter.

*Nothing is explicitly stated about the temperature used in these papers; we are making such conclusions by looking at their code implementations: Snippet from Lowe et al. [2017]; Snippet from Papoudakis et al. [2021]

This alternative estimator is the simplest change one can make, and serves as an easy comparison.

Temperature-Annealed Gumbel Softmax (TAGS)

A fundamental challenge in RL is the *exploitation-exploration dilemma* [Sutton and Barto, 2018], which describes the trade-off between taking actions that yield known, good rewards ('exploiting'), and taking actions which may or may not yield *better* rewards ('exploring'). In the original formulation of MADDPG [Lowe et al., 2017, Appx: Alg. 1], exploration is achieved via the addition of noise to the policy output: $a_i = \mu_i(o_i) + \eta_i$, where η is drawn from some random process (originally discussed in DDPG [Lillicrap et al., 2016]). However, this is applicable in *continuous*-action spaces. For discrete cases, the GS itself provides some degree of exploration, since relaxing the distribution places some probability mass onto other actions. Naturally, the amount of exploration is controlled by the temperature parameter—more relaxation implies more exploration.

Notice, then, the coupling between the exploration achieved and the bias introduced: both are affected by changes in the temperature. Ideally, this would not be the case, such that the hyperparameters could be tuned independently. Decoupling them would be a long-term solution—an avenue for future work—but in the short-term, a work-around must be made.

Since exploration is usually desirable in the *beginning* of a training procedure, we propose setting the temperature to be high early-on, and then annealing it to be lower over time. This allows agents to explore, while still reducing the bias in later stages of training. Huijben et al. [2022] highlight temperature-annealing as a strategy incorporated by several authors in various experiments with the GS. Specifically, they mention using an exponentially-decaying annealing scheme, which we adopt here. We define this as the Temperature-Annealed Gumbel-Softmax (TAGS) estimator.

Gumbel-Rao Monte Carlo (GRMC)

The next estimator for this project is entitled the GRMC, by Paulus et al. [2021]. Here, the authors build on the success of the GS, but seek a way to lower the estimator's *variance*. Returning to the Gumbel-Max trick, they note that the $\arg\max(\zeta_i + g_i)$ operation is *not* invertible, implying that many instances of $\zeta_i + g_i$ correspond to the same action selection. Accordingly, they view the drawn Gumbel random variables as

'auxiliary', which can be marginalised out.

Notating the gradient of the original STGS estimator as $\nabla_{\text{STGS}} = \nabla_a \xi_{\text{STGS}}$, we have:

$$\nabla_{\text{STGS}} := \frac{d\text{softmax}_\tau(\zeta + g)}{da} \tag{3.5}$$

With this notation, the authors propose the 'Gumbel-Rao' estimator:

$$\nabla_{\text{GR}} := \mathbb{E}\left[\frac{d\text{softmax}_\tau(\zeta + g)}{da} \,\middle|\, a\right] \tag{3.6}$$

That is, $\nabla_{\text{GR}} = \mathbb{E}[\nabla_{\text{STGS}} \mid a]$. This estimator is a 'Rao-Blackwell' [Blackwell, 1947] version of the original STGS estimator. It can be shown that it thus enjoys the same mean as the STGS, but with lower (or at most, the same) variance:

$$\mathbb{E}\left[||\nabla_{\text{GR}} - \nabla_\zeta||^2\right] \leq \mathbb{E}\left[||\nabla_{\text{STGS}} - \nabla_\zeta||^2\right] \tag{3.7}$$

where ∇_ζ is the 'true' gradient. For rigorous mathematical details about the estimator's impact on variance, the reader is encouraged to see the full paper [Paulus et al., 2021]. Recall, however, Papoudakis et al. [2021] refer to the *bias* of the estimator as the problem for MADDPG, not the *variance*. Though guarantees are only made about the latter, the authors argue that with a lower variance, one can safely train the estimator at lower temperatures—i.e. with a lower bias. Empirically, they show this to be true.

Though theoretically appealing, there is still the challenge of actually computing $\mathbb{E}[d\text{softmax}_\tau(\zeta + g)/da \mid a]$—indeed, a closed-form expression is shown to be difficult. Therefore, the authors provide a Monte Carlo estimate, with K samples, which they term the GRMCK estimator. They first show a distributional equivalence:

$$(\zeta_j + g_j \mid a) \overset{d}{=} \begin{cases} -\log(E_j) + \log Z(\zeta) & \text{if j = i} \\ -\log\left(\frac{E_j}{\exp(\zeta_j)} + \frac{E_i}{Z(\zeta)}\right) & \text{otherwise} \end{cases} \tag{3.8}$$

where a is a one-hot sample with a 1 at index i, E_j are independent and identically distributed (i.i.d.) samples from the exponential distribution, and $Z(\zeta) = \sum_j \exp(\zeta_j)$.

Accordingly, the GRMCK estimator is:

$$\nabla_{\text{GRMC}K} := \frac{1}{K}\sum_k^K \frac{d\text{softmax}_\tau(\zeta + g^k)}{da} \quad , \quad g^k \sim (\zeta + g \mid a) \tag{3.9}$$

In other words, we first sample $a \sim p(a;\zeta)$, and then average over K Gumbel samples *conditioned on* a. The result is an estimator with lower variance, which can thus be trained at lower temperatures, with a lower bias. As before, we show this procedure graphically on the probability simplex, in Figure 3.2.

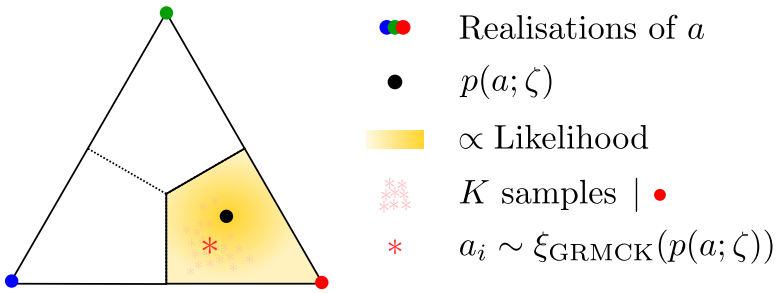


Figure 3.2: Depiction of the GRMC method on the probability simplex. We sample a as red in this example, and condition accordingly. We see that the K samples are thus drawn from this conditioned space, and when averaged, yield a lower-variance sample. (Adapted from Fan et al. [2022])

Gapped Straight-Through (GST)

The final estimator considered in this project is the most recently introduced: the GST, by Fan et al. [2022]. Also building on past work, these authors study both the STGS and the GRMC, and focus on their key properties: what is essential for good performance, and what is merely ancillary? In particular, they focus on the Gumbel perturbation employed in the STGS—finding that certain features of this perturbation are required in the estimator, but *not* the Gumbel randomness. Indeed, they show that rather using *deterministic* perturbations—two, specifically—satisfies the necessary requirements for estimation, while boasting lower variance.

As in the GRMC estimator, we first draw $a \sim p(a;\zeta)$—a one-hot representation of the selected action—for the straight-through sample. In GRMC, we would then perturb each of the logits, ζ, with Gumbel noise conditioned on a; now, we perturb with two deterministic functions, $m_1(\zeta,a)$ and $m_2(\zeta,a)$. The detailed justification for the choices of these functions is outside of the scope of this project; instead, the reader should see the exposition in the paper itself [Fan et al., 2022]. For now, we describe them at a high-level.

Firstly, we desire 'consistency' in the estimator: we want the sample conditioned on a to have the same largest logit as the input distribution, i.e. $\max_j \zeta_j$. To this end, the first perturbation, m_1, 'pushes' the sample to the correct realisation, if necessary:

$$m_1(\zeta,a) = \left(\max_j \zeta_j - \langle \zeta, a \rangle\right) \cdot a \tag{3.10}$$

where $\langle \cdot,\cdot \rangle$ indicates the inner product. Consider how this works: if a has already selected the largest logit, then $\langle \zeta, a \rangle = \max_j \zeta_j$, and $m_1 = 0$. If not, then $m_1 \neq 0$, and the sample is moved in the direction of the largest logit.

If non-zero, the first perturbation makes the largest logit the *same* as the a-selected logit. However, we also want a 'strict gap' between these values—that is, we want the *unselected* logits to be smaller. Accordingly, we define m_2 to create a gap of κ between them:

$$m_2(\zeta, a) = -\left(\kappa + \zeta - \max_j \zeta_j\right)_+ \odot (1-a) \tag{3.11}$$

where $(x)_+ := \max(0, x)$, $\odot$ indicates the Hadamard product. κ can usually be set to 1.0 [Fan et al., 2022]. Here, the term $(1-a)$ takes all the unselected logits in the one-hot representation, and moves their parameter values away from the selected logit, with a gap of at least κ.

With these perturbation functions defined, the GST estimator is then:

$$\xi_{\text{GST}}(p(a;\zeta)) = \text{softmax}_\tau(\zeta + m_1(\zeta, a) + m_2(\zeta, a)) \tag{3.12}$$

This procedure is again nicely visualised on the probability simplex. Figure 3.3 shows how the perturbations move the sample, given a particular realisation of a. The resulting estimator is shown to have lower variance [Fan et al., 2022], and as before, can thus be trained at lower temperatures, with lower bias.

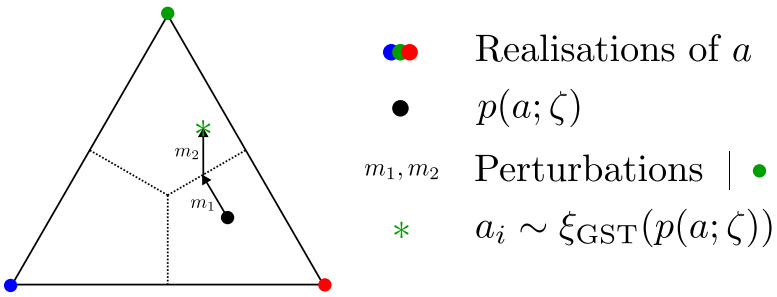


Figure 3.3: Depiction of the GST on the probability simplex. We sample a as green in this example. Notice how m_1 then moves the original logits, specified by ζ, to the 'border' of green and red; m_2 then enforces a 'strict gap'. (Adapted from Fan et al. [2022])

Chapter 4

Experimental Methods

This chapter provides a succinct outline of the methodology we use in the project's experiments. We discuss the environments in which we test, and the metrics with which we evaluate, as well as the details of the training—e.g. the hyperparameters used.

4.1 Environments

To test the performance of the proposed gradient estimators, compared to the original STGS estimator, we train on three 'environments', with multiple 'tasks' (i.e. configurations) for each environment—with a total of 11 tasks overall. We use a sensible subset of the choices made by Papoudakis et al. [2021] in their benchmarking paper, which we motivate shortly. For simplicity, we choose to focus solely on *co-operative* contexts, where agents are working together to maximise their cumulative reward—we feel this is sufficient to demonstrate the aims of the project. Adversarial environments are nonetheless important, and should form a part of future research.

The three implemented environments are described briefly below.

Multi-Agent Particle Environment (MPE)[*]: This environment was created by Mordatch and Abbeel [2018], with adaptations made by Lowe et al. [2017] when presenting MADDPG. In it, each agent is represented as a particle that can move around a continuous, two-dimensional space, while interacting with other agents and various 'landmark' items. There are a variety of navigational tasks defined, and agents receive *dense* reward signals for these tasks. The observation space for each agent usually contains some information about distances to other agents and landmarks, along with

[*]Code for MPE is here: https://github.com/semitable/multiagent-particle-envs

metrics like agent-velocity. Communication between agents is also defined for some tasks. Each agent has a discrete-action space of the four cardinal directions, and not moving at all.

Importantly, we highlight that in the benchmarking paper by Papoudakis et al. [2021], MADDPG performed fairly *well* in the MPE scenarios, similar to other MARL algorithms. Accordingly, the goal for this environment—and the motivation for its inclusion here—is *not* to improve on the returns with the alternative gradient estimation techniques. Instead, we hope to use it as a sanity check: a simple test that each of the alternative estimators has been implemented correctly and works with MADDPG. Of course, such a check does not *guarantee* correctness, but it is a good indicator. We test with two tasks from MPE: `speaker_listener`, and `spread`.

Level-Based Foraging (LBF)[*]: This environment was implemented for a paper by Christianos et al. [2020], building on work by Albrecht and Ramamoorthy [2013]. Tasks consist of a discrete grid-world, with a collection of agents and randomly-placed food. Each agent has a 'level', and likewise for each food item. An agent may collect food if the sum of the agents' levels directly adjacent to the food is at least as large as the food's level. That is, collection is successful iff: $\sum \text{level(adj. agents)} \geq \text{level(food)}$. These tasks are highly customisable, and a plethora of combinations exists across grid-size, agent-count, food-count, and observability (full or partial). An agent can act discretely across 6 options: no operation, the four cardinal directions, and the 'load' operation, where it tries to pick up food.

In LBF, agents are solely rewarded based on the food they collect, and therefore, rewards can be sparse—this makes it a challenging environment. Indeed, in the benchmarking paper [Papoudakis et al., 2021], MADDPG performed comparatively *poorly* in LBF, particularly in the more complex tasks (e.g. with larger grid-sizes). Accordingly, this environment is vital for testing the alternative gradient estimation techniques. The same seven configurations for LBF are implemented here as those in the work by Papoudakis et al. [2021]—covering three grid-sizes, various agent-food pairs, and some partial observability challenges. This provides a broad suite of tests on which to evaluate the project's aims.

Robot Warehouse (RWARE)[†]: This environment was also implemented by Christianos et al. [2020], based on work by Albrecht and Ramamoorthy [2016]. Here, a

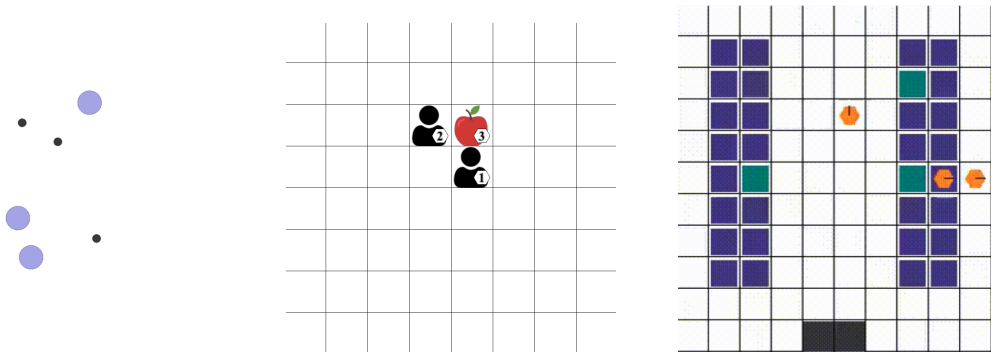
[*]Code for LBF is here: https://github.com/semitable/lb-foraging

[†]Code for RWARE is here: https://github.com/semitable/robotic-warehouse

task consists of robot agents, moving around a discrete grid-world warehouse. Agents must move goods from a shelf to a goal-location, based on given requests. Agents are rewarded for a successful delivery, though such rewards are very sparse—more so than for LBF. As a result, this is a very challenging environment, and MADDPG performed even worse in the benchmarks [Papoudakis et al., 2021] for RWARE than it did for LBF.

As an initial insight, then, we consider two of the RWARE tasks: `tiny-2ag` and `tiny-4ag` (i.e. the tiny grid-size, 10×11, with 2 or 4 agents). These should demonstrate whether there is some hope in trying MADDPG with alternative estimators in RWARE—success here would then encourage testing with other RWARE tasks in future.

Sample renderings of the three environments are given in Figure 4.1.



(a) MPE (b) LBF (c) RWARE

Figure 4.1: Example renderings of the three environments used for this project

4.2 Evaluation Metrics

To understand the success (or failure) of the new gradient estimation techniques compared to the original STGS approach, metrics for evaluation ought to be chosen. Specifically, we focus on two metrics for *performance*: the returns achieved when implemented with MADDPG, and the time required for computation, in a 'toy' gradient estimation problem. Moreover, we present a brief foray into trying to understand *why* a particular method might do well over the others, by looking at the gradient variance for a single task. We discuss these metrics below.

Returns: Maximum & Average

Recall that we define our MARL goal, in (2.13), as trying to find an optimal set of policies, such that each agent maximises their expected return with respect to the other agents' policies. The achieved return, then, is an important metric to measure. Since we focus on co-operative situations for this project, we simply consider the sum of the achieved returns from all agents. Importantly, we are not concerned with the returns achieved here relative to those achieved in, e.g., the MARL benchmarking paper by Papoudakis et al. [2021]. Instead, for cogent and consistent analysis, we focus solely on the relative performance of the various estimators against each other.

For this evaluation metric, we run the MADDPG algorithm in each task, with each of the proposed gradient estimators. We train the algorithm for a fixed number of time-steps, updating the networks with a defined period. Throughout training, we evaluate the achieved returns 100 times every 50 000 time-steps. Each training iteration is done over five random seeds, and from this, a 95% confidence interval is calculated for the results.

Under this heading, we consider two distinct aspects of the achieved returns, following the lead of Papoudakis et al. [2021]. Firstly, we consider the *maximum* return. For this, we find the evaluation time-step at which the return, averaged over the five seeds, is highest. This indicates the raw performance of the algorithm when using a given estimator.

Secondly, we consider the *average* return. For this, we average the evaluation returns over all time-steps and seeds, for a given estimator in a given task. This metric provides a proxy for understanding not just the magnitude of the returns, but how *quickly* the algorithm can arrive at such returns in its training—i.e. how quickly its training *converges*.

Compute Time

Though this project revolves around—and is motivated by—the MADDPG algorithm, notice that the gradient estimators can also be compared in isolation. That is, when comparing the computational burden of the various estimation procedures, we need not integrate them into the broader MADDPG problem. Instead, we can take a closer look solely at each estimator's performance, unhindered by potential bottlenecks elsewhere.

Accordingly, we define here a simple, 'toy' problem for the estimators. We define a set of input logits, ζ, of various dimensionalities, and measure the time it takes for

each estimator to calculate the corresponding relaxations. Because STGS-1, STGS-T, and TAGS all have the same underlying mechanics, we consider these under the single umbrella of the STGS. For the GRMCK, we consider three values of K: 1, 10, and 50. For each dimensionality, the estimation procedure is repeated 10 000 times, over five different logit instances. These results are reported over a 95% confidence interval.

Gradient Variance

Suppose one of the alternative gradient estimation techniques performs significantly better or worse than the original STGS, based on the returns achieved. The natural follow-up question is: why? For our investigation to be complete, it is helpful if we have empirical evidence for why a particular estimator is performing better than another. This is not the *primary* aim of this project, and we cannot dive deeply into this topic here—instead, we hope to provide an initial insight.

To do this, we choose one of the tasks where there is a notable difference in performance between two estimators: between the baseline STGS-1 method and one which performs much better (or worse). We then retrain the MADDPG algorithm in this task, using each of the two estimators, this time logging the *variance* of the computed gradients across each training mini-batch. We do this over the course of training, and focus on any differences observed between the two methods.

We hypothesise that *uninformative* gradients—i.e. those due to a poor discrete-gradient estimator—will yield a mini-batch with *low* variance, since there are no elements in particular which 'stand out'. In contrast, we believe that *informative* gradients will have higher variance across the mini-batch, for the opposite reason—particular components in the gradient vectors are more important than others, yielding large differences between them.

At present, this belief is merely a hunch, but this is sufficient for our purposes—we hope only to stimulate discussion into why we might observe a difference in estimator performance. There is a fertile ground here for future research.

4.3 Training Details

Hyperparameters

Hyperparameter tuning is often an important, though time-consuming, component of training RL algorithms. For simplicity, then, the optimal hyperparameters for the core

MADDPG algorithm suggested by Papoudakis et al. [2021] are adopted here, mostly without any changes. Table 4.1 reports the values used.

Table 4.1: Hyperparameters used for the core MADDPG algorithm, mostly taken verbatim from Papoudakis et al. [2021]

	MPE	LBF	RWARE
network type	MLP	MLP	MLP
hidden dimensions	(128,128)	(64,64)	(64,64)
learning rate	5e-4	3e-4	3e-4
reward standardisation	True	True	True
policy regulariser	0.001	0.001	0.001
target update (β)	0.01	0.01	0.01
max timesteps	25	25	500
training interval (steps)	25	25	50

This limits our hyperparameter search to be over the novel gradient estimation techniques and their associated parameters. Bayesian optimisation [Garnett, 2022] is performed for this search, and we use search-range suggestions from the literature, when available [Huijben et al., 2022, Paulus et al., 2021]. Each parameter is optimised for one task in a particular environment, and then used for all other tasks in that environment. The associated hyperparameter ranges for the estimators are described in Table 4.2, along with the selected values for each.

Table 4.2: Hyperparameter details for the various gradient estimation techniques, with the chosen parameters listed for the three environments.

Estimator:	Range Explored	MPE	LBF	RWARE
STGS-1	$\tau = 1.0$	1.0	1.0	1.0
STGS-T	$\tau \in (0,1)$	0.6	0.5	0.6
TAGS	$\tau \in [1,5] \rightarrow [0.1,0.5]$	$2.5 \rightarrow 0.5$	$4.0 \rightarrow 0.1$	$1.0 \rightarrow 0.3$
GRMCK	$\tau \in (0,1]; K = \{5,10,50\}$	1.0;5	0.5;10	0.7;5
GST	$\tau \in (0,1]$	0.6	0.7	0.7

Computational Load

Experiments for this project were run using single-core nodes on the University of Edinburgh's high-performance computing cluster, Eddie, and on Google Cloud's Compute Engine. Excluding the hyperparameter search, a total of 4952 CPU-hours were spent.

Chapter 5

Experimental Results & Discussion

This chapter presents the results of the experiments outlined previously, paired with brief, relevant discussions. We consider first the returns achieved in MADDPG with each gradient estimation technique, and then the required compute-time for each in a toy-problem; finally, we run a short experiment analysing the gradient variance in one of the tasks.

5.1 Returns: Maximum & Average

Table 5.1 shows the maximum and average returns achieved using each gradient estimation technique, across each of the 11 tasks. We discuss these results below, per environment, showing plots when relevant.

Multi-agent Particle Environment (MPE)

As mentioned previously, the MADDPG algorithm performed relatively well on MPE tasks in the benchmarks presented by Papoudakis et al. [2021], achieving similarly strong results to other MARL algorithms. Hence, we do not expect major improvements with the alternative estimators—we use this environment essentially as a sanity check.

Indeed, we see that each of the alternative techniques performs similarly to the baseline (STGS-1) for the two MPE environments, and any differences are statistically insignificant. For that reason, we move forwards with increased confidence in our implementation, and do not consider this environment further.

Table 5.1: Maximum returns (Average returns) shown across all tasks and all algorithms, presented with a 95% confidence interval over 5 seeds. **Bold** indicates the best performing metric for a situation. An asterisk ($*$) indicates that a given metric is *not* significantly different from the best performing metric in that situation, based on a heteroscedastic, two-sided t-test with 5% significance. Under each task name is the number of time-steps used for training.

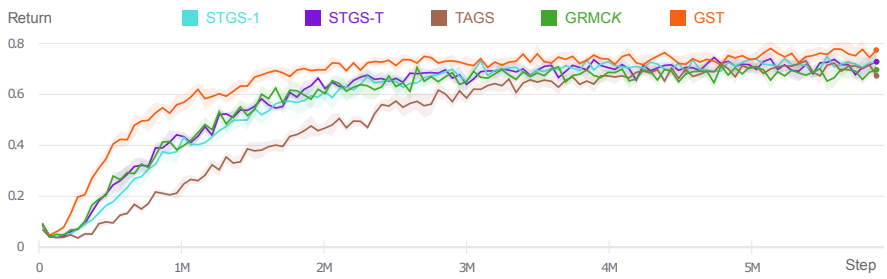
	Tasks	STGS-1	STGS-T	TAGS	GRMC*K*	GST
MPE	speaker-listener	$-13.03 \pm 1.29^*$	$-13.72 \pm 1.55^*$	$\mathbf{-12.88 \pm 1.83}$	$-13.69 \pm 0.89^*$	$-16.05 \pm 3.53^*$
	[2M]	$(\mathbf{-18.52 \pm 3.57})$	(-19.33 ± 3.53)	(-19.01 ± 3.71)	(-19.86 ± 3.53)	(-21.89 ± 3.22)
	spread	$-133.70 \pm 2.03^*$	$-133.52 \pm 2.78^*$	$-134.66 \pm 3.27^*$	$\mathbf{-133.49 \pm 2.06}$	$-135.67 \pm 1.94^*$
	[2M]	$(\mathbf{-145.91 \pm 5.55})$	(-147.07 ± 6.08)	(-147.49 ± 6.11)	(-146.20 ± 5.79)	(-147.81 ± 5.82)
Foraging	8x8-2p-2f-c	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.01}$	$\mathbf{1.00 \pm 0.01}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
	[5M]	(0.88 ± 0.05)	$(\mathbf{0.91 \pm 0.04})$	(0.87 ± 0.05)	(0.88 ± 0.05)	(0.89 ± 0.04)
	8x8-2p-2f-2s-c	$0.79 \pm 0.07^*$	$\mathbf{0.83 \pm 0.03}$	$0.78 \pm 0.03^*$	$0.81 \pm 0.05^*$	$0.81 \pm 0.02^*$
	[5M]	(0.65 ± 0.04)	(0.66 ± 0.04)	(0.62 ± 0.04)	(0.67 ± 0.04)	$(\mathbf{0.68 \pm 0.03})$
	10x10-3p-3f	0.75 ± 0.03	0.75 ± 0.03	0.74 ± 0.06	0.71 ± 0.07	$\mathbf{0.79 \pm 0.04}$
	[6M]	(0.58 ± 0.04)	(0.59 ± 0.03)	(0.51 ± 0.04)	(0.57 ± 0.03)	$(\mathbf{0.66 \pm 0.03})$
	10x10-3p-3f-2s	$0.55 \pm 0.05^*$	$\mathbf{0.58 \pm 0.06}$	0.54 ± 0.03	$0.56 \pm 0.03^*$	$0.56 \pm 0.05^*$
	[6M]	(0.48 ± 0.01)	(0.48 ± 0.01)	(0.46 ± 0.01)	(0.49 ± 0.01)	$(\mathbf{0.50 \pm 0.01})$
	15x15-3p-5f	0.24 ± 0.02	0.28 ± 0.06	0.20 ± 0.03	0.26 ± 0.05	$\mathbf{0.31 \pm 0.04}$
	[7.5M]	(0.12 ± 0.01)	(0.15 ± 0.01)	(0.08 ± 0.01)	(0.16 ± 0.01)	$(\mathbf{0.20 \pm 0.01})$
	15x15-4p-3f	0.79 ± 0.03	0.79 ± 0.06	0.77 ± 0.06	0.79 ± 0.04	$\mathbf{0.83 \pm 0.04}$
	[7.5M]	(0.54 ± 0.04)	(0.58 ± 0.04)	(0.45 ± 0.04)	(0.58 ± 0.04)	$(\mathbf{0.67 \pm 0.03})$
	15x15-4p-5f	0.33 ± 0.06	0.46 ± 0.12	0.24 ± 0.05	0.43 ± 0.06	$\mathbf{0.48 \pm 0.06}$
	[7.5M]	(0.13 ± 0.02)	(0.22 ± 0.02)	(0.10 ± 0.01)	(0.21 ± 0.02)	$(\mathbf{0.30 \pm 0.02})$
RWARE	tiny 2ag	$1.37 \pm 0.22^*$	$1.37 \pm 0.49^*$	$\mathbf{1.50 \pm 0.46}$	$1.37 \pm 0.40^*$	$1.40 \pm 0.58^*$
	[7.5M]	(0.55 ± 0.07)	(0.64 ± 0.08)	(0.60 ± 0.08)	$(\mathbf{0.65 \pm 0.07})$	$(\mathbf{0.65 \pm 0.07})$
	tiny 4ag	2.68 ± 0.49	3.18 ± 0.60	2.23 ± 0.50	3.17 ± 0.85	$\mathbf{4.16 \pm 0.97}$
	[7.5M]	(0.84 ± 0.12)	(1.09 ± 0.15)	(0.78 ± 0.11)	(1.15 ± 0.15)	$(\mathbf{1.82 \pm 0.21})$

Level-Based Foraging

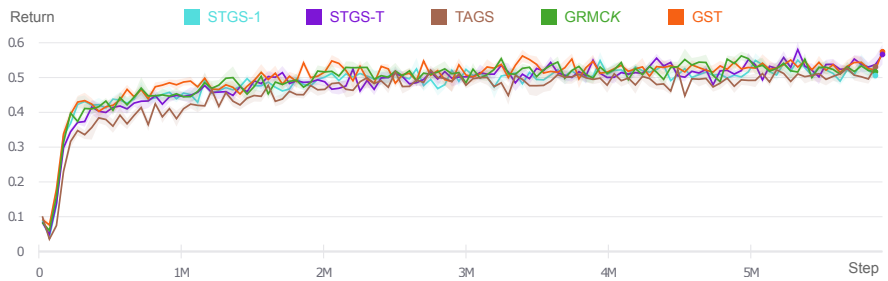
We consider now the LBF environment across seven different tasks.

Firstly, we look at two tasks with an 8×8 grid: one with full-observability (`8x8-2p-2f`), and one with partial-observability (`8x8-2p-2f-2s`). Notice in these results, in Table 5.1, that performance differences across the estimation techniques is statistically insignificant. In `8x8-2p-2f`, we see that STGS-T trains marginally *faster* than the other approaches, and in `8x8-2p-2f-2s`, we see that TAGS trains marginally *slower* than the other approaches—both based on the average returns observed. Nonetheless, each algorithm arrives at a similar maximum return. Due to the insignificance of this result, the training curves are uninteresting, and are not plotted here.

We next look at two tasks with an 10×10 grid, with similar situations as before: one with full-observability (`10x10-3p-3f`), and one with partial-observability (`10x10-3p-3f-2s`). Plots of the evaluation returns over the duration of training are given in Figure 5.1.



(a) `lbf-10x10-3p-3f`



(b) `lbf-10x10-3p-3f-2s`

Figure 5.1: Evaluation returns for two LBF tasks (10×10) over the training period, where the shaded region indicates the standard error as calculated over 5 seeds.

In the first task, seen in Figure 5.1a, we see an improvement with a novel gradient estimation technique: the GST achieves the highest maximum and average returns

for the task, beating the baseline STGS-1 method with statistical significance. It is clear in the figure how training with the GST converges faster than with the other methods. STGS-T and GRMC*K* perform similarly to the baseline. TAGS, however, performs much *worse* in average returns—i.e. it converges slower for the task—though it eventually achieves similar maximum returns.

In the task with partial observability, seen in Figure 5.1b, we are less successful—the alternative techniques achieve statistically similar returns to the baseline, across both maximum and average metrics. The exception is TAGS, which again performs *worse* than the baseline, though not markedly so.

We consider now the remaining three tasks in LBF, with a fully-observable, 15×15 grid: `15x15-3p-5f`, `15x15-4p-3f`, and `15x15-4p-5f`. We recall that MADDPG performed particularly poorly in these larger, more-complex LBF situations, according to the benchmarking paper by Papoudakis et al. [2021]. The training curves for each of these tasks is given in Figure 5.2.

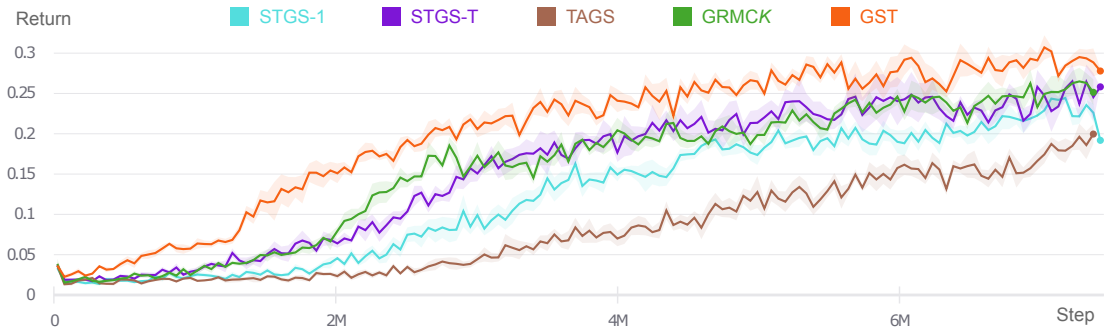
We notice here significant improvements over the baseline. Yet again, TAGS markedly underperforms, both in maximum and average returns; but the other estimators perform well. STGS-T and GRMC*K* beat the baseline in average returns for `15x15-3p-5f` and `15x15-4p-3f`, and in both average and maximum returns for `15x15-4p-5f`. GST is superior throughout: across all three tasks, it yields significantly higher returns and converges faster than the baseline (and the other techniques). Indeed, these improvements are clearly noticeable in the plots provided.

Robot Warehouse (RWARE)

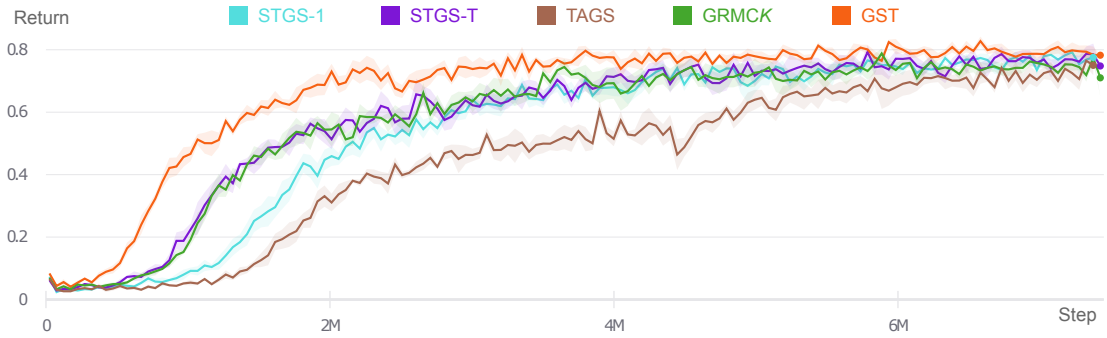
Finally, we consider the RWARE environment for two tasks: `tiny-2ag` and `tiny-4ag`. Figure 5.3 shows the returns for these two environments, over the training period.

In `tiny-2ag`, we see insignificant differences across the estimation techniques, with each achieving similar maximum returns. The alternative techniques do converge slightly faster, particularly GRMC*K* and GST, with marginally higher average returns, but not by much.

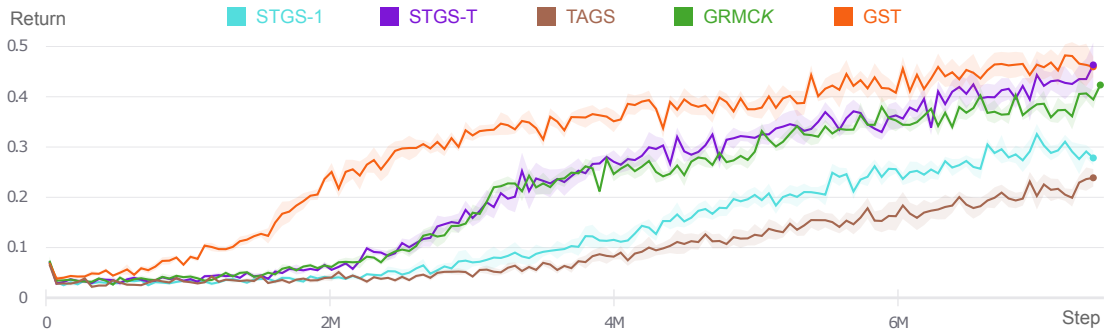
In `tiny-4ag`, we see the most significant improvements yet. Barring TAGS, which somewhat underperforms, we notice substantial improvements from the other proposed estimators, for both average and maximum returns. GST triumphs once more, achieving 1.5 times the maximum returns of the baseline, and over 2 times the average returns. This is again clear in the plot, in Figure 5.3b.



(a) `lbf-15x15-3p-5f`

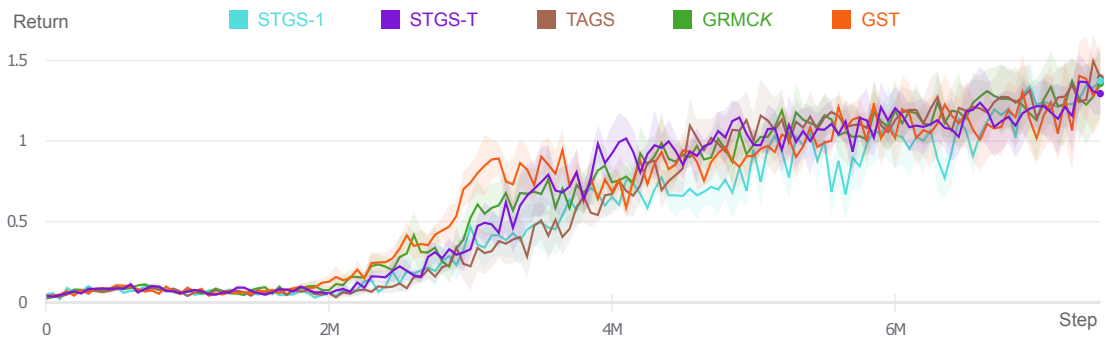


(b) `lbf-15x15-4p-3f`

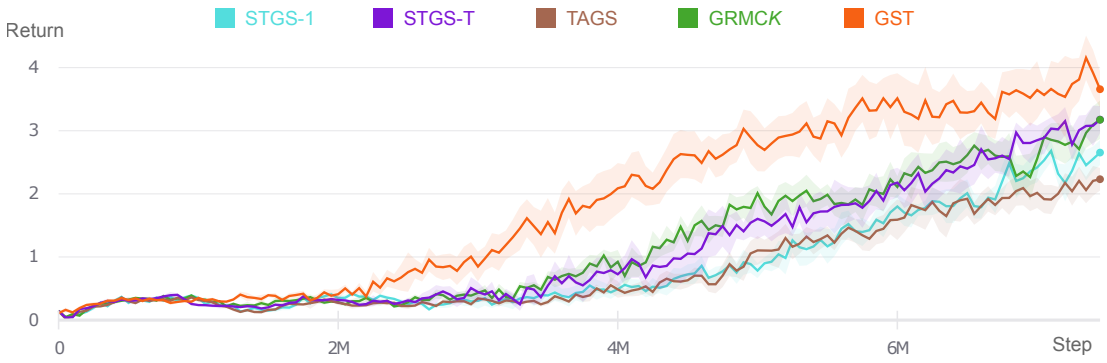


(c) `lbf-15x15-4p-5f`

Figure 5.2: Evaluation returns for three LBF tasks (15×15) over the training period, where the shaded region indicates the standard error as calculated over 5 seeds.



(a) `rware-tiny-2ag`



(b) `rware-tiny-4ag`

Figure 5.3: Evaluation returns for two RWARE tasks (tiny grid) over the training period, where the shaded region indicates the standard error as calculated over 5 seeds.

Discussion

This section presented the results from training MADDPG with each of the proposed gradient estimation techniques, across 11 tasks from three environments. In some tasks, particularly the more straightforward ones (e.g. MPE, or LBF with 8×8 grid-size), the alternative techniques do not make a significant difference to the returns achieved. We suspect this is because informative gradients are not as crucial in simple tasks. That is, the gradient estimation is not a problematic aspect of the training, and limitations arise elsewhere in the mechanics of MADDPG.

Interestingly, in some of the more challenging tasks, particularly in LBF with a grid-size of 15×15 and `tiny-4ag` in RWARE, we see significant improvements. We note that simply lowering the temperature (and hence, the gradient estimator's bias), as in STGS-T, can improve the results somewhat—supporting the hypothesis that the bias introduced by the GS is a problem for MADDPG. The Rao-Blackwellisation procedure of GRMC*K* also sees better returns and faster convergence. Much better than these,

though, is the GST. With this estimator, we consistently see marked improvements across the two return metrics, and these are statistically significant.

Though a lower temperature seems to yield better returns, our results suggest that *annealing* the temperature, as in TAGS, performs poorly. This may be due to the coupling of exploration and exploitation, as highlighted earlier, but more investigation is required. Alternatively, these bad results may simply be the hyperparameters chosen—the annealing start and end points were taken from the advice of Huijben et al. [2022]. It is conceivable that using *lower* values here may yield better returns, especially considering the improvements seen with STGS-T. Future work could also explore using alternative annealing schemes, or annealing with a different underlying estimator—e.g. one could try a temperature-annealed *GST*.

From these results, considering both the maximum returns achieved and the time to convergence, we note that alternative gradient estimation techniques can indeed yield better returns when incorporated into MADDPG—particularly the recently-proposed GST, from Fan et al. [2022].

5.2 Compute Time

We now consider the computational requirements for each of the algorithms, using the toy-problem outlined earlier. Recall that we perform these tests for three classes of estimator: STGS (which accounts for STGS-1, STGS-T, and TAGS); GRMC (with three different K values); and GST. Table 5.2 shows the outcome of these tests.

We notice firstly that STGS scales well with dimensionality—the computational overhead when increasing the dimension does not change significantly. Even in the high-dimensional case of 1 000, the technique is only marginally slower. These benefits are common to the baseline STGS-1 approach, as well as the proposed techniques of STGS-T and TAGS.

Next, we see that GRMCK is at least three times slower than the baseline approach. Moreover, using a larger K value does, understandably, increase the computational burden of the relaxation. Though this is not substantial for low-dimensional inputs, for higher-dimensional problemsx, K has a marked impact—e.g. with $K = 50$, computation slows down considerably, becoming 40 times slower than the baseline for an input dimension of 1 000.

The computational burden of GST sits somewhat in-between the baseline, STGS, and the GRMCK approach. Importantly, though, this method also scales well with

Table 5.2: Time-per-relaxation, in μs, for the three classes of gradient estimators, when using logits of various dimensionality as input. Results are given over a 95% confidence interval from 5 different logit instances, where each procedure is repeated 10 000 times. Underneath each metric, using round brackets, $(\cdot)$, we indicate how much slower the alternative techniques are, when compared to the baseline STGS.

Estimator → ↓ Dimensionality	**STGS**	**GRMC-1**	**GRMC-10**	**GRMC-50**	**GST**
3	135.28 ± 0.19 (1.0)	445.91 ± 11.38 (3.3)	446.65 ± 0.86 (3.3)	486.36 ± 1.7 (3.6)	357.56 ± 1.04 (2.64)
5	135.65 ± 0.51 (1.0)	438.52 ± 0.63 (3.23)	446.66 ± 0.7 (3.29)	501.9 ± 1.16 (3.7)	356.95 ± 0.6 (2.63)
10	135.83 ± 0.48 (1.0)	438.95 ± 1.01 (3.23)	451.65 ± 0.82 (3.33)	531.97 ± 0.45 (3.92)	356.52 ± 0.55 (2.62)
50	134.05 ± 0.89 (1.0)	440.46 ± 1.01 (3.29)	484.77 ± 1.29 (3.62)	765.56 ± 2.09 (5.71)	356.77 ± 1.77 (2.66)
100	139.23 ± 0.25 (1.0)	455.54 ± 2.97 (3.27)	520.06 ± 0.78 (3.74)	1055.18 ± 4.16 (7.58)	359.26 ± 1.26 (2.58)
1000	154.17 ± 0.45 (1.0)	533.12 ± 1.01 (3.46)	1060.74 ± 2.92 (6.88)	6171.59 ± 8.67 (40.03)	386.57 ± 1.36 (2.51)

dimensionality, staying at just over 2.5 times slower than the baseline, irrespective of the input size. This is an attractive property.

From these results, and the insights drawn from the previous section, we can draft general guidelines for choosing an alternative estimator: if minimising the computational burden is paramount for a given problem, it may be worth using the STGS-T, for it has the same overhead as the STGS-1, and it does yield improvements in both the achieved returns and convergence time. However, if one can afford a more expensive relaxation procedure, the GST is a good fit—it is somewhat slower, but the benefits are significant. Since GRMCK is more expensive than the GST, yet usually yields lower returns, it does not seem like a sensible option as an estimator in either case. Granted, it could be that better performance comes from increasing K further (e.g. Paulus et al. [2021] use $K = 1000$ in some of their experiments), but the computational burden will only worsen in such a case, and this is undesirable. TAGS, as it was presented here, should not be used.

5.3 Gradient Variance

We have previously seen marked improvements in some tasks when using the proposed gradient estimators, particularly the GST. We now stimulate further discussion by presenting a cursory look into *why*. For this, we reconsider the LBF task of `15x15-4p-5f`, and retrain with two algorithms: the baseline STGS-1, and the best performing alternative, GST. Figure 5.4 shows the variance of the gradients across mini-batches, for each of the layers in the policy networks, over the course of the training.

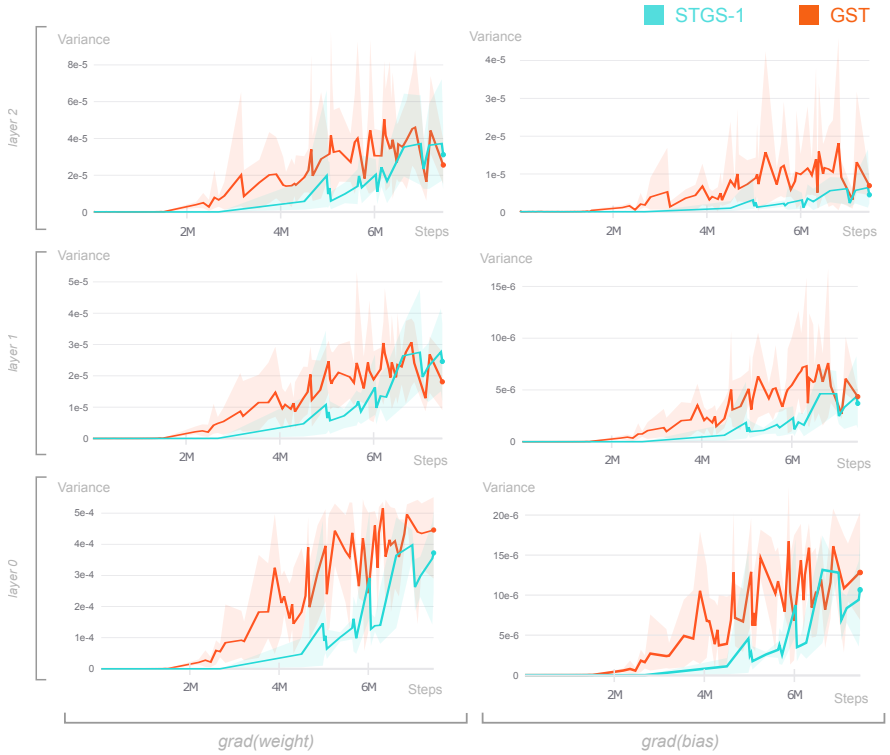


Figure 5.4: Plots showing the gradient variance (left: gradients of *weight* parameters, right: gradients of *bias* parameters), for each layer in the policy networks, for the `15x15-4p-5f` task in LBF. The results are aggregated across the 4 agents in this task—the shaded region indicates the maximum and minimum values across agents, the solid line indicates the mean.

Immediately, we notice a trend in these graphs—the variance of the gradients, taken across a mini-batch, increases more rapidly for the GST algorithm than those for the baseline. As argued earlier, though not definitive, such results indicate that more informative gradients are being propagated through the policy networks. Informative gradients, in turn, allow the algorithm to achieve higher returns and converge faster—as evidenced in Figure 5.2c.

Chapter 6

Conclusion

6.1 Summary

This project explored the impact of the Gumbel-Softmax (GS) reparameterisation on MADDPG when applied to discrete-action environments. This was done by first presenting the necessary theoretical foundations, and framing the problem in the context of the broader literature. Thereafter, we looked closely at the GS itself, and discrete gradient estimation more generally, highlighting the key concepts therein. After synthesising a handful of candidate GS alternatives, a select-few methods were implemented into the MADDPG algorithm. These methods were tested on a suite of 11 MARL tasks across three environments, and a variety of metrics were analysed.

On some of the tasks—particularly the simpler ones, where MADDPG already performed well—no significant changes were observed, in terms of returns achieved and the speed to convergence. On other tasks though, particularly in the more challenging ones, substantial improvements occurred. It was found that even an easy change to the original GS estimator—simply lowering the temperature parameter—yielded good results. The proposed temperature-annealing scheme in TAGS, however, was shown to be a bad choice for the estimator—though we acknowledge a different set of hyperparameters may have helped here. The GRMC estimator, presented by Paulus et al. [2021] showed promising results, but was hindered by a below-par computational burden. Finally, far superior to the other methods, was the GST estimator, presented by Fan et al. [2022]. This method achieved the best results across a range of tasks, with higher returns and faster convergence, when compared to the original GS. Though it did introduce additional computational burden, the method nonetheless scaled well with dimensionality, and is certainly a viable technique for many use-cases.

We are now in a good position to support the suggestions made by Papoudakis et al. [2021] in their benchmarking paper. Based on the empirical data observed, we agree that the bias of the GS method is indeed problematic for MADDPG. As a result, by improving the estimator used—i.e. by lowering its bias—we can improve the returns achieved by MADDPG. To answer our original question, then: yes, alternative discrete gradient-estimation techniques *can* improve the performance of MADDPG in discrete grid-worlds.

Notice the benefit of such an outcome. We can take the extant MADDPG algorithm, replace *only* the gradient estimation technique—that is, swap out, e.g., the GS for the GST, and leave everything else the same—and the resulting performance may likely improve. Though our algorithm becomes slightly more expensive computationally, we witness faster convergence and higher returns, with minimal development overhead.

6.2 Future Work

Many avenues of future work extend from this project—we highlight a handful here.

Firstly, it would be useful to evaluate the performance of the proposed algorithms on a wider variety of tasks—in particular, the results in RWARE were promising but distinctly limited, with only *two* tasks tested. Moreover, only co-operative tasks were tested here, and adversarial configurations should be explored too.

Secondly, much more investigation ought to be done into *why* the alternative techniques are boasting better performance. Though an interesting foray, the analysis into the gradient variance was just a first step. Future research should continue to focus on the mechanics of the algorithms, and probe at various points.

Thirdly, the core MADDPG algorithm designed for this project did not incorporate various extensions suggested in the literature, e.g. parameter sharing. Combining the benefits observed here with other strong extensions elsewhere would be an interesting exercise. Furthermore, a wider hyperparameter search, now with the alternative estimators involved too, may be helpful.

Finally, we note that only two alternative methods from the literature were presented here—the GRMC and the GST. Though sufficient for our analysis, it would be useful to explore the other options synthesised from the literature. Some of these, though far more complex, boast many attractive properties, and may prove to be even more fruitful.

This project aimed to be a doorway for future investigation, and based on the results observed, it seems to be an inviting one.

Bibliography

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 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1155–1156, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450319935.

Stefano V. Albrecht and Subramanian Ramamoorthy. Exploiting causality for selective belief filtering in dynamic bayesian networks. *Journal of Artificial Intelligence Research*, 55:1135–1178, 2016.

Evgeny Andriyash, Arash Vahdat, and Bill Macready. Improved Gradient-Based Optimization Over Discrete Distributions. *arXiv:1810.00116 [cs, stat]*, June 2019.

David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

David Blackwell. Conditional expectation and unbiased sequential estimation. *The Annals of Mathematical Statistics*, pages 105–110, 1947.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 10707–10717. Curran Associates, Inc., 2020.

Xiangxiang Chu and Hangjun Ye. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*, 2017.

Morris H. DeGroot and Mark J. Schervish. *Probability and Statistics International Edition*. Pearson Education, Limited, 2011. ISBN 9780321709707.

Alek Dimitriev and Mingyuan Zhou. CARMS: Categorical-Antithetic-REINFORCE Multi-Sample Gradient Estimator. In *Advances in Neural Information Processing Systems*, volume 34, pages 13217–13229. Curran Associates, Inc., 2021.

Wei Du and Shifei Ding. A survey on multi-agent deep reinforcement learning: From the perspective of challenges and applications. *Artificial Intelligence Review*, 54(5):3215–3238, June 2021. ISSN 0269-2821, 1573-7462. doi: 10.1007/s10462-020-09938-y.

Ting-Han Fan, Ta-Chung Chi, Alexander I. Rudnicky, and Peter J Ramadge. Training discrete deep generative models via gapped straight-through estimator. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6059–6073. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/fan22a.html.

Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. in preparation.

Peter W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, oct 1990. doi: 10.1145/84537.84552.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Duvenaud. Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. *ICLR 2018*, 2018.

Evan Greensmith, Peter Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper/2001/file/584b98aac2dddf59ee2cf19ca4ccb75e-Paper.pdf.

Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05176.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International conference on autonomous agents and multiagent systems*, pages 66–83. Springer, 2017.

Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, page 709–715. AAAI Press, 2004. ISBN 0262511835.

Iris A.M. Huijben, Wouter Kool, Max Benedikt Paulus, and Ruud JG Van Sloun. A Review of the Gumbel-max Trick and its Extensions for Discrete Stochasticity in Machine Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3157042.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. *ICLR 2017*, 2017.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, may 1996. doi: 10.1613/jair.301.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

Jack Kleijnen and Reuven Y. Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996. URL https://EconPapers.repec.org/RePEc:eee:ejores:v:88:y:1996:i:3:p:413-427.

Wouter Kool, Herke van Hoof, and Max Welling. Estimating Gradients for Discrete Random Variables by Sampling without Replacement. *ICLR 2020*, 2020.

Matt J. Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution, 2016. URL https://arxiv.org/abs/1611.04051.

Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization Gradient for Non-differentiable Models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR 2016*, January 2016.

Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization - an empirical study on continuous control. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=yr1mzrH3IC.

Guy Lorberbom, Andreea Gane, Tommi Jaakkola, and Tamir Hazan. Direct Optimization through argmax for Discrete Variational Auto-Encoder. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Ryan Lowe, Wu Yi, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. *Advances in neural information processing systems*, 27, 2014.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *ICLR 2017*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Wokrshop*, December 2013.

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 *Proceedings of The 33rd International Conference on Machine Learning*, pages 1928–1937. PMLR, June 2016.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo Gradient Estimation in Machine Learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020. ISSN 1533-7928.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL probml.ai.

Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V. Albrecht. Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning. *arXiv:1906.04737 [cs, stat]*, June 2019.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 1, December 2021.

Max B. Paulus, Chris J. Maddison, and Andreas Krause. Rao-Blackwellizing the Straight-Through Gumbel-Softmax Gradient Estimator. *ICLR 2021*, 2021.

Adeel Pervez, Taco Cohen, and Efstratios Gavves. Low bias low variance gradient estimates for boolean stochastic networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7632–7640. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/pervez20a.html.

Andres Potapczynski, Gabriel Loaiza-Ganem, and John P Cunningham. Invertible Gaussian Reparameterization: Revisiting the Gumbel-Softmax. In *Advances in Neural Information Processing Systems*, volume 33, pages 12311–12321. Curran Associates, Inc., 2020.

Jason Tyler Rolfe. Discrete Variational Autoencoders. *ICLR 2017*, 2017.

Julien Roy, Paul Barde, Félix Harvey, Derek Nowrouzezahrai, and Chris Pal. Promoting coordination through policy regularization in multi-agent deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:15774–15785, 2020.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-020-03051-4.

L. S. Shapley. Stochastic Games. *Proceedings of the National Academy of Sciences*, 39 (10):1095–1100, October 1953. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.39.10.1095.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395. PMLR, January 2014.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel,

and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature16961.

Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Andreas Veit and Serge J. Belongie. Convolutional networks with adaptive inference graphs. *International Journal of Computer Vision*, 128:730–741, 2019.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-019-1724-z.

M. Waltz and K. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398, oct 1965. doi: 10.1109/tac.1965.1098193.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3): 279–292, 1992.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 538–545, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608001.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.

Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, February 2022. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-021-04357-7.

Shiyang Yan, Jeremy S Smith, Wenjin Lu, and Bailing Zhang. Hierarchical multi-scale attention networks for action recognition. *Signal Processing: Image Communication*, 61:73–84, 2018.

Shangtong Zhang, Hengshuai Yao, and Shimon Whiteson. Breaking the deadly triad with a target network. In *International Conference on Machine Learning*, pages 12621–12631. PMLR, 2021.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4006–4015. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/zhang17b.html.