Coordination, Detection and Collaboration in Deep Multi-Agent Reinforcement Learning

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

by

Chapman Siu

to

Faculty of Engineering and Information Technology
University of Technology Sydney

June 2023

Certificate of Original Authorship

I, *Chapman Siu* declare that this thesis, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the *School of Electrical and Data Engineering*, *Faculty of Engineering and Information Technology* at the University of Technology Sydney, Australia. This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

Production Note:
Signature removed prior to publication.

SIGNATURE: _____

[Chapman Siu]

DATE: 30th June, 2023

PLACE: Sydney, Australia

Abstract

Numerous real-world problems including controlling autonomous self-driving cars, drones, and robot swarm control are typically formulated as a cooperative *multi-agent reinforcement learning* (MARL) challenge. In cooperative MARL, one key challenge is handling the enormous joint action space. This joint action space increases at an exponential rate relative to the agents present in the MARL task. For instance, if there are six agents which have nine actions to select from, this will yield a space representing the joint actions which exceeds over ten million possibilities[1]. Effective MARL methods must be capable of operating over a large joint action space, particularly when the application of traditional single-agent reinforcement learning methods are not practically viable.

In the first section, we examine the usage of dynamic coordination graphs for MARL called *Dynamic Q-value Coordination Graph* (QCGraph). QCGraph aims to dynamically represent and generalise through factorising the combined value function of all agents. This is achieved by using global information generated from analysing subsets of agents. This allows agents to learn coordination even when conducting decentralised execution.

The second part of the thesis investigates challenges around detection of distribution shift in reinforcement learning which occurs when attempting to use offline experiences which may be sub-optimal. Specifically we explore *Dual Behavioural regularised Actor Critic* (DBR) to train agents to gravitate towards "good" experiences, whilst avoiding "bad" ones. This is achieved through constructing two behavioural policies to model "good"

[1] six agents with nine actions contains $6^9 > 10000000$

and “bad” experiences and constraining the resulting learning process of the policy to ensure the distributional shift is bounded. Although we can apply the general concepts in DBR to multi-agent reinforcement learning, we can also use experiences of other agents. We explore this in *Multi-Agent regularised Q-learning* where we examine mechanisms for correcting for the differences in experiences across agents based on their underlying policy distribution when sharing experiences.

In the final part of the thesis we explore the task of collaboration, where we allow agents to behave in ways which purposely neglect the coordination aspect of mixing networks which are commonly used in MARL algorithms. To examine this, we first explore neural network ensemble approaches which can learn disjointly and provide theoretical guarantees in the boosting framework. We use this ensemble learning approach in MARL in an approach we call *Greedy UnMixing* which allows us to combine the mixer and individual networks such that the mixing network can be ‚Äúunmixed‚Äù, thereby allowing agents to become uncoordinated explicitly.

List of Publications

Conference Papers

- Siu, C., 2019. Residual Networks Behave Like Boosting Algorithms. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (pp. 31-40). IEEE.

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. Dynamic Coordination Graph for Cooperative Multi-Agent Reinforcement Learning. To appear in *Asian Conference on Machine Learning 2021*.

Preprints

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. Dual Behavior regularised Reinforcement Learning. *arXiv preprint arXiv:2109.09037*.

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. regularise! Don't Mix: Multi-Agent Reinforcement Learning without Explicit Centralised Structures. *arXiv preprint arXiv:2109.09038*.

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. Greedy UnMixing for Q-Learning in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2109.09034*.

List of Figures

Figure **Page**

List of Tables

LaTeX Font Warning: Some font shapes were not available, defaults substituted.

LaTeX Warning: There were multiply-defined labels.

1

Introduction

Contents

1.1 Introduction

Historically, artificial intelligence was focused on knowledge based reasoning and expert systems. However these approaches had limitations for tasks that most humans find straight-forward such as image recognition tasks. This necessitated the advancement of machine learning and pattern recognition. These approaches focused on providing a probabilistic approach, which introduced algorithms such as linear regression, gradient boosted trees, and many other algorithms. Deep learning has further shown tremendous

progress primarily in the fields of image and pattern recognition, as well as natural language understanding, which have been driven through large datasets and the usage of graphical processing units. In all the examples listed previously, it is assumed that supervised learning is used, where the training dataset is assumed to be independent of the decisions made by the algorithms.

However in reinforcement learning, this assumption no longer holds. Instead algorithms take actions which influence the *state* of the environment. Then within the reinforcement learning setup, an agent (e.g. an autonomous car), interacts with the environment (e.g. the act of driving in a city) by taking actions based on what it perceives through its sensors, including steering, acceleration and braking. At every point of time, an agent observes and collects rewards from the environment, and acts accordingly.

An agent's strategy or actions based on observations is known as the 'policy' and reinforcement learning typically aims to maximise the cumulative discounted reward. Rewards are discounted over a time horizon. This forces rewards earned in the far future to be less influential than rewards received near-term. This enables agents to correctly assign value to actions. We define a trajectory as the progression of observations, rewards, and actions that end on a terminal condition, which can be time-based, or when an agent completes (or fails) the task. In reinforcement learning, actions are stateful, where the actions can change the immediate reward, or the transition probability relating to the next state, and/or future rewards. Our thesis is focused on model-free scenario, where the agent is not presented with the rules governing the reinforcement learning task, and therefore is required to learn based on past trajectories of experiences through interacting with the environment.

1.2 Overview

Reinforcement Learning (RL) typically tackle problems that involves a single agent to solve a single task. However, many challenges involve multiple learning agents, leading to Multi-Agent Reinforcement Learning (MARL) algorithms. One of the central challenges is handling the complexity surrounding the enormous joint action space in the multi-agent setting. As an example, six agents each having nine actions to select from, will yield a space representing the joint actions exceeding over one hundred million actions. This demonstrates the exponential growth relative to the increase in number of agents. As a consequence, MARL methods need to be able to generalise over this complex joint action space, as the application of traditional single-agent reinforcement learning methods are impractical. Throughout this thesis, we address MARL challenges using the commonly used approach called Centralised Training with Decentralised Execution (CTDE). As part of this framework, training is carried out in a centralised manner, in which the simulator or environment may have access to global state information, such as global information on other agents. However in the evaluation or execution phase, the agents are restricted in their observation and are required to act independently based only on local state or local observations only.

We focus on developing novel deep MARL algorithms in this thesis. In particular we focus on the following three facets: *coordination*, *detection*, and *collaboration*. *Coordination* explores how we can dynamically induce coordination graphs to learn and exploit graph representations of agents to improve policies. *Collaboration* describes the challenge of exploiting the multi-agent paradigm to enable efficient learning through decoupling single and multi-agent state spaces in the MARL framework. Finally, *detection* describes the challenge that arises in accounting for changes in the distributional shift in agent experiences when leveraging suboptimal experience, or accounting for distributional shifts when leveraging other agent experiences.

1.3 Multi-Agent Reinforcement Learning

Research in RL is typically focused around single agent approaches. However, real world challenges often necessitates environments or tasks that contain a large number of autonomous, learning agents. This leads towards the need for *multi-agent* RL. There are numerous examples of these tasks, including autonomous vehicles, drones and packet delivery, among other examples. These tasks typically require agents to make decisions based on their own local environment-based observations. This may include other agents, which could be biological in nature such as humans or animals. The reason why MARL algorithms are required is to explore approaches to constructing effective policies, algorithms, and rules in these settings. Part of the challenge when applying single agent RL approaches to multi-agent tasks is the increase in the joint action space complexity. For example, if we apply single agent reinforcement learning to a task which involves eight agents with nine actions each, it would lead to attempting to optimise a joint action space over 100 million in size, rendering this approach impractical.

The MARL field aims to address these kinds of problems, and seeks to develop and analyse algorithms that can discover and create effective policies in these settings. As humans we have the ability to learn from negative experiences, whether it is to actively avoid making the same mistakes again or through observing the mistakes of others. In this thesis we address how agents can acquire a similar ability through the challenge of *detection* - can an agent detect bad experiences and learn from them? Can an agent learn from other agent's experiences and apply it to their context where appropriate? A key component of reinforcement learning is for agents to explore efficiently. As we typically have multiple agents in multi-agent tasks, we address the challenge of *collaboration* by understanding how we can leverage multiple agents to provide diverse experiences through allowing agents to be decoupled from centralised structures. In the real world, cooperative tasks can be complex. Sometimes it is advantageous to act alone, or in

pairs or as a collective. We address the challenge of *coordination* to understand how centralised training structures can be dynamically altered to cater for these scenarios.

These observation culminate in the three cardinal components of this thesis: *detection*, *collaboration* and *coordination* within the cooperative MARL framework. We hypothesise that further consideration around sharing agent experiences and the *detection* of distributional shifts can be used to optimise MARL policies when there are limited or sub-optimal data. This can allow for flexible neural network structures in which agents can ignore centralised mixing networks in MARL algorithms, enabling greater diversity during exploration, and the ability to generate diverse representations through graph structures can assist in *coordination*.

To address these challenges, we will take advantage of *centralised training, decentralised execution* framework. This framework is commonly used in the real world where agents are to act or execute autonomously, based on its own local observation, but may be trained in a closed environment such as a simulator or a laboratory. Centralised training can greatly enhance the learning process in multi-agent problems and allow algorithms to exploit the central state information without requiring it upon evaluation.

1.4 Thesis Structure and Contributions

This thesis is concerned with Multi-Agent Reinforcement Learning. The methods which are introduced surround the three domains below, which are *learning to coordinate*, *learning to detect*, and *learning to collaborate*.

Below, we provide a brief explanation of each section.

Background

In Chapter 2 we formally introduce multi-agent reinforcement setting and provide the necessary background to deep multi-agent RL which is explored throughout this thesis.

Part I: Learning to Coordinate

The previous methods have not directly introduced new structures to the MARL framework; rather they have only introduced mechanisms to leverage existing experiences better and have multi-agents explore more effectively. To remedy how we can learn policies more effectively, we leverage ideas around coordination graphs. Having coordination graphs which are dynamically induced and learned can enable the mixer networks in the centralised training step to better represent the problem space and update the agents' policies. *Dynamic Coordination Graph* (QCGraph) is an approach which explores how this could be achieved and is competitive with other value factorisation methods in the Q-value MARL space.

This part draws on the following publications:

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. Dynamic Coordination Graph for Cooperative Multi-Agent Reinforcement Learning. To appear in *Asian Conference on Machine Learning 2021*.

Part II: Learning to Detect

One of the differences with single agent algorithms when transitioning towards algorithms in the MARL setting is the availability of different agent experiences that can be used for training all agents. One of the challenges is understanding and detecting distributional shifts in experiences across diverse sets of experiences that may be from the same or different agents and adjusting appropriately.

In Chapter 4 we explore *Dual Behaviour Regularised Reinforcement Learning* (DBR). DBR proposes an algorithm to learn from experiences which may be sub-optimal in nature through learning dual behavioural policies to model the distribution of "good" and "bad" experiences and constraining the policy updates to actively avoid "bad" experiences and move towards "good" experiences. This leverages the ideas discussed in two

papers, the "Advantage-based Regret Minimisation" framework [35], and "Self-Imitation Learning" through bootstrapping off an agent's past experiences [52]. This strategy has demonstrated the ability to:

- Constrain the exploration of an agent to avoid states leading to poor outcomes
- Constrain the policy evaluation under the regret minimisation framework
- Provide greater representational power compared with approaches which do not leverage "bad" experiences as a constraint

This approach was used in the MARL context when considering SEAC frameworks to enable efficient learning from past agent's experiences in for multi-agent environments with positive effect.

In Chapter 5 we introduce *Multi-Agent Regularised Q- Learning* (MARQ). This approach is different compared with existing MARL approaches, which leverage centralised structures to promote communication constraints and other global information., The latter often become extraneous and are removed at execution. Instead of explicitly learning redundant structures in the form of neural networks which are removed during agent execution phase in the CTDE framework, we propose to learn these structures implicitly. This is achieved through sharing agent experiences and the inclusion of a regularisation penalty which enables structured exploration. In this chapter, we propose two different ways in which shared experienced can be leveraged either implicitly or explicitly to regularise our policies in the MARL setting. Our approach addresses limitations in applying shared experiences naively through correcting bias when sharing agent experiences using off-policy methods through the addition of regularisation constraints. We observed empirically that this enables more efficient training as we require less data compared with other MARL approaches which leverage explicit centralised structures that lead to faster convergence and training speed. We explore how this can be achieved

through correcting for shared experiences and the inclusion of regularisation penalties to account for differences in policy distribution across multiple agents.

These chapters are based on the following pre-prints:

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. Dual Behaviour regularised Reinforcement Learning. *arXiv preprint arXiv:2109.09037*.

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. regularise! Don't Mix: Multi-Agent Regularised Q- Learning. *arXiv preprint arXiv:2109.09038*.

Part III: Learning to Collaborate

So far we have only considered the scenarios to leverage and optimise the set of experiences which the agents have collected, without consideration for how agents can effectively decouple their own experiences from other agent's experiences or trajectories.

In Chapter 6 we explore supervised learning techniques in *Residual Networks* and how its boosting representation can evolve into new and novel structures to combine disparate supervised learning models together. We extend this approach to MARL to demonstrate how we can use ensemble learning techniques to improve the performance of MARL algorithms with minimal changes to the neural network architecture. In Chapter 7 we introduce *Greedy UnMixing* (GUM) for Q-Learning in MARL tasks. GUM presents a novel ensembling approach to "unmix" or decouple the relationship between each agent's policy and the centralised mixing network. This simplifies the problem space, and is achieved through the "centralised training decentralised execution" framework, and allows the potential of agents to act independently (i.e. unmixing) from the centralised structures that are typically used in MARL frameworks.

This part draws on the following pre-prints and publications:

- Siu, C., 2019, October. Residual Networks Behave Like Boosting Algorithms. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (pp. 31-40). IEEE.

- Siu, C., Traish, J. and Da Xu, R.Y., 2021. Greedy UnMixing for Q-Learning in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2109.09034*.

2

Background

Contents

In this background chapter, an overview of approaches is provided related to reinforcement learning and other background information which are relevant to the remainder of the thesis. Concepts which are required only for specific chapters are introduced as additional background in those chapters. This chapter first introduces single-agent RL approaches in the Q-learning and Actor-Critic space, before addressing the corresponding

MARL variations.

2.1 Reinforcement Learning

First we introduce the key concepts related to reinforcement learning [65]. In reinforcement learning, we begin with an agent-environment framework [59] in which an agent progressively acts on and with the environment over time, denoted by sequential time steps, t by performing actions, collecting rewards and observations. The environment consists of state s_t from state space S at a particular timestep t, and is characterised by a transition probability function $P(s_{t+1}|s_t,u_t) : S \times U \times S \to \mathbb{R}$, the reward is defined as $R(s_t,u_t) : S \times U \to \mathbb{R}$, based on actions u_t from action space $u_t \in U$, with discount factor γ. This formulation is typically described as the Markov Decision Process (MDP), because the transition probability is conditional on the previous state-action pair. For fully observable MDP settings, an agent would observes the complete state of the task or environment s_t, and it would choose an action $u_t \in U$ according to the agent's policy $\pi(u_t|s) : S \times U \to [0,1]$. According the executed action, the environment would (either in deterministic or stochastic fashion) transition according to the transition probability $s_t \sim P(s_{t+1}|s_t,u_t)$, emiting a reward $r_t \sim R(s_t,u_t)$ to the agent. In this thesis, we only consider the model-free setting, in which the reward function and transition function are hidden, requiring agents to infer the dynamics of the respective task through its interactions with it, in order to construct the transition and reward functions. In reinforcement learning, we aim to maximise the total discounted reward according to a learned policy. The total discounted reward per episode is determined by a trajectory $\tau = \{s_0,u_0,r_0,\ldots,s_{T+1}\}$, which allows us to formally define the discounted reward as $J = \mathbb{E}_{\tau\sim P(\tau)}R_0(\tau)$. In this setup, $R_t(\tau) = \sum_{t'=t}\gamma_{t'}^{t'-t}$ is the forward-looking return from time step t onwards.

2.2 Deep Q-Learning

We begin our exploration of reinforcement learning algorithms by covering Q-learning and several variations in the Deep RL space.

Q-Learning. The Q function is commonly used and can be evaluated for any arbitrary policy π. To define this, we first consider the discounted return for any agent a for an episode to be $R_t^a(\boldsymbol{\tau}) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}^a$. Then the Q function (which is also known as the action-value function) is as follows:

$$Q_\pi^a(s_t, u_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}} \left(R_t^a | s_t, u_t \right) \tag{2.1}$$

We can also define the expectation over a given state which induces the value function:

$$V_\pi^a(s_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}} \left(R_t^a | s_t \right) \tag{2.2}$$

Then $Q^*(s,u) = \max_\pi Q^\pi(s,u)$ which defines the optimal action-value function follows the Bellman optimality equation shown below:

$$Q^*(s,u) = \mathbb{E}_{s'} \left(r + \gamma \max_{u'} Q^*(s', u' | s, u) \right) \tag{2.3}$$

This equation forms the underlying Q-learning approach. In approaches which do not leverage deep learning, Q-learning is typically resolved through environments which consist of discrete actions allowing for tabular approaches to be used to determine the corresponding Q values. One of the limitations of the tabular approach is that it operates on a discrete action space.

Deep Q-Networks. In deep Q-networks (DQN) [29, 48], rather than a tabular Q function, a deep neural network function $Q(s,u;\theta)$ is used, where θ represents the parameter for the function approximation characterised by a deep neural network. The objective function for DQNs at iteration i which is optimised through minimising the Bellman error shown below

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,u,r,s'}\left((y_i - Q(s,u;\theta_i))^2\right) \tag{2.4}$$

where the objective is $y_i = r + \gamma \max_{u'} Q(s',u';\theta_i^-)$, with θ^- being the parameter of a fixed deep neural network. This neural network is kept constant for several predetermined training epochs whilst updating the network $Q(s,u;\theta_i)$, and which is then periodically cloned with the baseline Q function.

DQN approach can be extended to "Double DQN" [29] which trains two deep neural networks with identical networks jointly, and leveraging the alternative network in the Bellman update equation.

Typically, a strategy called ϵ-greedy is a strategy used to select actions to promote diverse experiences and prevent overfitting. This procedure operates by permitting the agent to choose a random action during learning phase with probability ϵ and the greedy action that maximises the Q-value with probability $1-\epsilon$ to ensure agents continually explore during the RL task during training. DQNs also employ experience replay to enhance the learning process, whereby the agent builds a dataset of experiences and samples mini-batches during the training phase. This approach prevents overfitting due to recent experiences.

Distributional Q-Learning. Rather than only estimating the expected return as described in Q-learning and DQN approaches, distribution Q learning [2, 12] advocates modelling the distribution of returns, that is we aim to model the probability law of $R_i^a(\boldsymbol{\tau})$. To achieve this, one approach is to leverage quantile regression. This allows us to model the return distribution [12], so compared with the DQN scenario the output of the neural network is instead increased by an order of N, that represents the number of quantile targets, $\phi_1,\ldots,\phi_N$. Thus instead of mean squared error, the quantile Huber loss is used as shown below:

$$\mathcal{L}_i(\theta_i) = \sum_{i=1}^{N} \mathbb{E}_{s,u,r,s'} \left(\rho_{\phi_i}(y_i - Q(s,u;\theta_i)) \right) \tag{2.5}$$

where ρ_ϕ is the quantile Huber loss at quantile ϕ,

$$\rho_\phi(z) = \begin{cases} |\phi - \delta_{\{z<0\}}| \frac{1}{2} z^2, & \text{if } |z| \leq 1 \\ |\phi - \delta_{\{z<0\}}| \left(|z| - \frac{1}{2} \right), & \text{if } |z| \leq 1 \end{cases} \tag{2.6}$$

with δ indicating Dirac delta function.

Deep Recurrent Q-Networks. In partially observable settings as s_t contains global state information, it cannot be directly used as input for the agents. Instead, an agent a only receives observation o_t^a which would be correlated with s_t. Deep recurrent Q-networks [30] addresses this in single-agent, through approximating $Q(\tau, u)$ instead of $Q(s,u)$ with recurrent neural networks which maintains an internal state over time. This approach can then be used in partially observable settings. This is accomplished through using the hidden state of network as an additional input h_{t-1} and output h_t, and is used over a variety MARL algorithms to address the partially observability aspect.

2.3 Policy Gradient and Actor-Critic Approaches

In this section we examine policy gradient and actor-critic approaches. These methods differ from Q-learning approaches in that they require estimating a value function (which can be the Q function, V function, or even the reward itself) and using this value function to optimise a policy. In Q-learning approaches the policy is derived directly from the value function whereas in policy gradient and actor-critic approaches, the policy update procedure modifies the policy's parameters directly.

Policy Gradient. Before introducing actor-critic, we first consider policy gradient methods. These methods directly optimise an agent's policy, parameterised by θ^π as oppose to examining the Q function of a given state. The policy gradient objective is to

optimise the expected total discounted reward $J = \mathbb{E}_\pi(R_t)$. This results in the gradient in the form $\nabla J(\theta^\pi) = \mathbb{E}_\pi(\nabla_{\theta^\pi} \log \pi(u_t|s_t) Q(s_t, u_t))$ [66], in which different approaches could be used to estimate the value function Q. As an example, in REINFORCE [71] the form of this is provided as $\nabla J(\theta^\pi) = \mathbb{E}_\pi\left(\sum_{j=0}^{T} R_j \nabla_{\theta^\pi} \log \pi(u_j|s_j)\right)$. This approach suffers from high variance in the gradient estimation. This may mean that the algorithm requires more update steps compared with an approach with lower variance.

Actor-Critic. To remediate the limitations of REINFORCE, actor-critic methods were introduced, where the *actor* (or policy) is trained by following a gradient that is constructed through training the *critic*, which typically follows a variation of the value function. For example, one possibility is to replace R_t by the value function $V_(s_t;\theta^c)$, where the critic network is parameterised by θ^c. Then the critic would aim to minimise the following loss:

$$\mathcal{L}(\theta^c) = \left(V(s_t;\theta^c) - y\right)^2 \quad \text{with } y = r + \gamma V_i(s_{t+1};\theta^c) \tag{2.7}$$

Actor-critic methods could leverage the usage of a target network, in order to keep the parameters of y constant for a number of iterations, whilst training the neural network, or leverage polyak averaging [27].

Deep Deterministic Policy Gradient. As Q-learning approaches generally only operate in the discrete space, approaches like Deep Deterministic Policy Gradient (DDPG) can be used which follow the spirit of DQN, but operates in continuous action spaces. This approach extends policy gradient methods to deterministic policies $\mu : S \rightarrow U$, so that the gradient of the objective is modified to $\nabla J(\theta^\pi) = \mathbb{E}\left(\nabla_{\theta^\pi} \mu(u|s;\theta^\pi) \nabla_u Q^\mu(s,u)|_{u=\mu(s)}\right)$. DDPG leverages deep neural networks as a function approximator for the policy and critic, and can leverage ideas from DQN such as replay buffers and target networks [44].

2.4 Multi-Agent Reinforcement Learning

In our work we examine the Multi-Agent Reinforcement Learning (MARL) problem. One of the key challenges of MARL is adapting to the enormous joint action space. A decentralised, partial observable Markov Decision Process [53] is formulated through the tuple which comprises of $\langle S, U, P, R, Z, O, n, \gamma \rangle$. We define the environmental state to be described by $s_t \in S$ where $t = 1, 2, \ldots$ is the timestamp as dictated by the environment. Then for every time step t, the agents $a \in A \equiv \{1, \ldots, n\}$, from which they select from $u \in U$ to act accordingly. These actions form the joint action defined by $\mathbf{u} \in \mathbf{U} \equiv U^n$. Similar to Q-learning, we define the probability transition function in the environment to be the state transition function: $P(s_{t+1}|s_t, \mathbf{u}_t) : S \times \mathbf{U} \times S \rightarrow [0, 1]$. All agents share $R(s_t, \mathbf{u}_t) : S \times \mathbf{U} \rightarrow \mathbb{R}$ which describes our reward function, and $\gamma \in [0, 1)$ which is the discount factor.

2.4.1 Partial Observability

Typically, in MARL environments, agents do not have full state information. This is formalised by restricting each agent to their own local, individual observation $z \in Z$ based on observation function we define as $O(s, a) : S \times A \rightarrow Z$. Using this formulation, each agent would each have their own action-observation history $\tau^a \in T \equiv (Z \times U)^*$. This is conditioned on each agent's policy $\pi^a(u^a|\tau^a) : T \times U \rightarrow [0, 1]$. This could be interpreted as the probability distribution over the agent actions. In this thesis we restrict our discussion to only consider the global, shared reward scenario, and agents are unable to directly observe an individual reward signal. We primarily focus on cooperative, partially observable multi-agent problems.

2.4.2 Centralised Training Decentralised Execution

To tackle large action spaces, a common approach is via decentralising the agent's value function and/or decision policy, whereby an agent's policy is restricted only to the agent's own local action-observation history. To facilitate training and communication amongst agents, learning may occur in a centralised fashion to provide additional state information and to remove communication constraints among agents. This framework is known as Centralised Training with Decentralised Execution (CTDE) [40, 54]. In this thesis, we focus on this paradigm, which means that centralised training may utilise extra information (such as access to additional state information or communication between agents, as long as this information is not available or accessible to the policy during execution time. This approach is the standard approach for solving Decentralised Partially Observable Markov Decision Process (Dec-POMDP) [53] and is used in numerous experiments provided in this thesis. This paradigm is used in our thesis as it best represents real-world constraints whereby agents are required to interact with their environment independently. Constructing agents in this manner ensure that it is robust to scenarios where we may presume joint policy across multiple agents such as in the centralised training and centralised execution paradigm. Finally there is also the decentralised training and decentralised execution paradigm where, where each agent only updates its own policy with no external knowledge or information, which is more useful in scenarios where agents are heterogenous and adversarial. In our thesis we focus on cooperate tasks which lend itself better for Centralised Training with Decentralised Execution (CTDE) approaches.

2.4.3 Q-Value factorisation Approaches to MARL

In this section, we discuss modifications to DQN that enables supporting multi-agent and partially observable settings. Applying a single-agent approach like DQN naively to

a multi-agent setting would require finding the $\text{argmax}_{\mathbf{u}}$ over the enormous action space which makes this impractical. Rather than computing the $\text{argmax}_{\mathbf{u}}$ in multi-agent tasks, many algorithms address this challenge through factorisation in the centralised training phase known as the "Individual Global Max Condition" (IGM) [5, 6, 57, 63].

Definition 2.1. IGM is defined as a joint-action function for n agents $Q_{\text{joint}} : \{\tau^1, \ldots, \tau^n\} \times \{u^1, \ldots, u^n\} \to \mathbb{R}$ where $\{\tau^1, \ldots, \tau^n\}$ is the joint-action histories for agents $1, \ldots, n$, if individual Q state action functions exists where $[Q^i : T \times U \to \mathbb{R}]_{i=1}^n$, and the following holds

$$\text{arg}\max_{\mathbf{u}} Q_{\text{joint}}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \text{arg}\max_{u^1} Q^1(\tau^1, u^1) \\ \vdots \\ \text{arg}\max_{u^N} Q^n(\tau^n, u^n) \end{pmatrix}$$

then, we say that $[Q_i]$ satisfy **IGM** for Q_{joint} under τ. In this case we also say that Q_{joint} is factorised by $[Q_i(\tau_i, u_i)]$ or that $[Q_i]$ are factors of Q_{joint}.

Independent Q-Learning. The simplest and perhaps most widely leveraged algorithm in MARL is the Independent Q-Learning (IQL) [67] approach. This approach forgoes centralised training altogether, and let each agent a learn greedily based solely on their own observed space to yield the action-value function Q^a independently. This approach, by definition cannot represent interaction between agents and consequently does not have convergence guarantees. This is due to the fact that each agents' learning is confounded by the environment and the activity of other agents. Despite this, IQL has a strong empirical record [75] and continues to be a relevant baseline for many MARL tasks.

Value Decomposition Networks (VDN) [64] is one factorisation which satisfies the IGM condition by being framed as the sum of independent agent's Q state action functions, Q^a. The factorised Q function for VDN is then defined to be

$$Q_{\text{joint}}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q^i(\tau^i, u^i) \tag{2.8}$$

Through the additive nature of VDN, this joint action value function would satisfy the IGM condition.

QMIX. Another approach which attempts to factorise Q_{joint} is to leverage a mixer network to combine each agent's state action function. The key differences compared with VDN is the ability to include global state information as part of the centralised training which would be excluded during decentralised execution. To combine each agent's state action function Q^a and the global state information s, QMIX allows for any family of monotonic functions such that the constraint between Q_{joint} and each Q^a is:

$$\frac{\partial Q_{\text{joint}}(\boldsymbol{\tau}, \mathbf{u}, s)}{\partial Q^i(\tau^i, u^i)} \geq 0 \ , \quad \forall i \in A \tag{2.9}$$

This monotonic constraint is enforced by constructing a mixing network where all the weights are non-negative. This has been shown to approximate any monotonic function [13].

QTRAN. Whilst the previous approaches is restricted to only combining utility functions through additive manner (VDN) or through monotonic mixing network (QMIX), QTRAN aims to generalise this to any action-value function [63]. This is constructed through applying linear constraints on the agent state action functions on the objective function. To complete this, three value functions are defined, Q_{joint}, which is any hypernetwork to combine each individual agent's state action function, Q_{vdn}, which represents the sum of each individual agent's state action function as per the VDN formulation, and $V_{\text{qtran}}(\boldsymbol{\tau}) = \max_{\mathbf{u}} Q_{\text{joint}}(\boldsymbol{\tau}, \cdot) - \max_{\mathbf{u}} Q_{\text{vdn}}(\boldsymbol{\tau}, \cdot)$. Then enforcing the constraint $Q_{\text{vdn}} - Q_{\text{joint}} + V_{\text{qtran}} \geq 0$ during the optimisation step will guarantee that the IGM condition is maintained. This is estimated through calculating the optimal local action which we denote by Q' that is determined counterfactually. That is, given $Q'^{\,i}_{\text{joint}}(\boldsymbol{\tau}, \{u^i \cup$

$\mathbf{u}^{-i}\};\theta)$, $\mathbf{u}^{-i} := (u^1,\ldots,u^{i-1},u^{i+1},\ldots,u^n)$, with $Q'_{\text{joint}}(\boldsymbol{\tau},\mathbf{u};\theta) = \sum_{i=1}^{n} Q'^{\,i}_{\text{joint}}(\boldsymbol{\tau},\{u^i \cup \mathbf{u}^{-i}\};\theta)$, where the fixed action vector $\mathbf{u}$ is determined by $\text{argmax}_{\mathbf{u}} Q_{\text{vdn}}(\boldsymbol{\tau},\cdot)$.

In addition to Equation 2.4, QTRAN calculates the loss relative to the "optimal" action and "non-optimal" action, which are combined in a weighted fashion that are hyperparameters to the QTRAN construction. The Losses relative to the "optimal" action and "non-optimal" action are shown in Equation 2.10 and 2.11 respectively:

$$\left(Q'_{\text{joint}}(\boldsymbol{\tau},\mathbf{u};\theta^-) - \hat{Q}_{\text{joint}}(\boldsymbol{\tau},\mathbf{u};\theta^-) + V_{\text{qtran}}(\boldsymbol{\tau};\theta)\right)^2 \tag{2.10}$$

$$\left(\min\left(Q'_{\text{joint}}(\boldsymbol{\tau},\mathbf{u};\theta) - \hat{Q}_{\text{joint}}(\boldsymbol{\tau},\mathbf{u};\theta) + V_{\text{qtran}}(\boldsymbol{\tau};\theta),\ \ 0\right)\right)^2 \tag{2.11}$$

Where Q' is constructed through counterfactually determining the optimal action, and $\hat{Q}$ leverages a 'detach' operator to stop the gradient flow through the network.

2.4.4 Policy Gradient and Actor-Critic Approaches to MARL

Although this thesis does not focus on actor-critic multi-agent methods, we discuss various actor-critic MARL approaches as a point of comparison.

Independent Actor-Critic. In Independent Actor-Critic (IAC) allows each agent to possess their own policy network and value function. It leverages their own local action-observation history as input to the policy. To improve convergence and training speed, this method commonly uses parameter sharing between the policy network and value function.

Shared Experience Actor-Critic. Shared Experience Actor-Critic (SEAC) [9] is an extension of IAC whereby it applies experience sharing by combining gradients of different agents as part of its training step. This approach treats trajectories from other agents to be *off-policy* data, and is corrected through importance sampling. That is if $\lambda \in [0,1]$ is the weight of sharing, then the losses for the policy network and value network respectively for agent a become:

$$\mathcal{L}^{\text{seac}}(\theta^{\pi,a}) = \mathcal{L}^{a}(o_t^a;\theta^{\pi,a}) + \lambda \sum_{k\neq a} \frac{\pi(u_t^k|o_t^k;\theta^{\pi,a})}{\pi(u_t^k|o_t^k;\theta^{\pi,k})} \mathcal{L}^{a}(o_t^k;\theta^{\pi,a}) \tag{2.12}$$

$$\mathcal{L}^{\text{seac}}(\theta^{c,a}) = \mathcal{L}^{a}(o_t^a;\theta^{c,a}) + \lambda \sum_{k\neq a} \frac{\pi(u_t^k|o_t^k;\theta^{\pi,a})}{\pi(u_t^k|o_t^k;\theta^{\pi,k})} \mathcal{L}^{a}(o_t^k;\theta^{c,a}) \tag{2.13}$$

Where θ^{π},a, θ^{c},a is used to describe the parameters of the policy network and critic network related to agent a respectively and $\mathcal{L}(o;\theta)$ symbolises the loss calculates given observation o with parameter θ.

Multi-Agent Deep Deterministic Policy Gradient. Another way to extend IAC approach is described in Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [45]. This is achieved by maintaining a separate centralised critic networks. That is, each agent's critic network has direct access to all agents' policy information for centralised training purposes.

Learning Implicit Credit Assignment. In the previous approaches each agent had their own actor and critic networks. Learning Implicit Credit Assignment algorithm (LICA) [76] differs from these approaches as it leverages a centralised critic which is shared for all agents formulated as a hypernetwork. It introduces *entropy regularisation* to improve policy optimisation which induces more stochastic actions. Entropy regularisation has been shown to empirically improve policy optimisation through the addition of an entropy regularisation penalty. For example, modifying the training objective is one approach to explicitly achieve this [26, 27, 60]. Alternatively one could implicitly enable this through augmenting the reward function [70]. The policy objective has an additional entropy regularisation term in the form $\mathbb{E}_a\left(\mathcal{H}(\pi^a(\cdot|o_t^a)\right)$. In the LICA formulation the entropy calculation is normalised across all agents. The optimal joint action can be generated i nthis formulation through enabling consistent exploration across all agents.

Part I

Learning to Coordinate

Abstract

In this part we discuss how coordination is explored within graph structures with applications to Reinforcement Learning (RL). First we explore the usage of graphs to determine coordination among agents in the multi-agent reinforcement learning framework. Our method uses dynamically generated graph structures at every step in the environment which are created independent from the environment definition. Coordination in this setting is only defined in centralised training step. Even when agents are executed in a decentralised manner, we demonstrate this approach still maintains coordination, exceeding other current RL approaches in the decentralised execution centralised training paradigm.

3

Q-Learning with Coordination Graphs

Contents

3.1 Introduction

Within cooperative *multi-agent reinforcement learning* (MARL), one of the central challenges is tacking the enormous joint action space. For example, some of the environments used in this chapter have over eight agents with nine actions to select from, yielding a joint action space which exceeds over a million actions. This joint action space grows exponentially relative to the number of agents.

To tackle the large action space challenge, one approach is to decentralise the execution of agent decisions, whereby an agent policy is conditioned on only their local action-observation history. To facilitate training and communication amongst agents, learning may occur centrally with additional global knowledge, which implicitly provides latent information to agents as we remove the ability for agents to communicate amongst each other. This approach is known as Centralised Training with Decentralised Execution (CTDE) [40, 54]. Recently, these approaches have generally focused on *factored Q-value functions*, which have demonstrated success [57, 64], and which focus on the value function factorisation.

There has also been a focus on representational ability when these factorisation techniques suffer from structural constraints, that is it presumes fixed, predetermined relationships among the agents. This has been addressed through considering the joint action-value space within the centralised training step [5, 6, 63]. However, these approaches face challenges when adapting to a dynamic environment, where agent relationships may change as it interacts with the reinforcement learning environment. For example, the approach taken in [6] relies on constructing a new network or model for every pair of agents which becomes unmanageable when the agent size is large. In the scenario for [63] it directly factorises the full joint action-value function, where as

[5] suggests to use a static predetermined coordination graph with parameter sharing to alleviate the challenges presented within [6]. Both [5] and [63] presuppose a fixed, immutable coordination graph which may not necessarily be appropriate depending on the MARL problem.

3.1.1 Contribution

To address these challenges, we introduce Dynamic Q-value Coordination Graph (QCGraph). QCGraph represents the value function as a coordination graph (CG) with payoffs determined by dynamically generated subsets of coordinating agents. To achieve scalability, QCGraph employs hierarchical graph neural network techniques to enable parameter sharing among arbitrary dynamically generated CGs and the respective agent utilities at both a micro and macro level of relationships or coordination. Parameter sharing is commonly used within factorised MARL approaches. Within Value Decomposition Networks (VDN) [64] and Monotonic factorisation Value factorisation (QMIX) [57], parameters are shared across all utility or state action functions by conditioning only on an agent's history (i.e. previous local states and actions). Approaches like DCG [5] and QTRAN [63] take this to the extreme approximating *all* payoff functions with the same function approximator. For QCGraph, we extend the limitations of DCG to allow for dynamic CG, as opposed to having a predefined, fixed CG, and is trained using deep Q-learning approaches. The QCGraph's performance is compared with other MARL Q-learning algorithms and actor-critic algorithms in a variety of tasks which require coordinated actions.

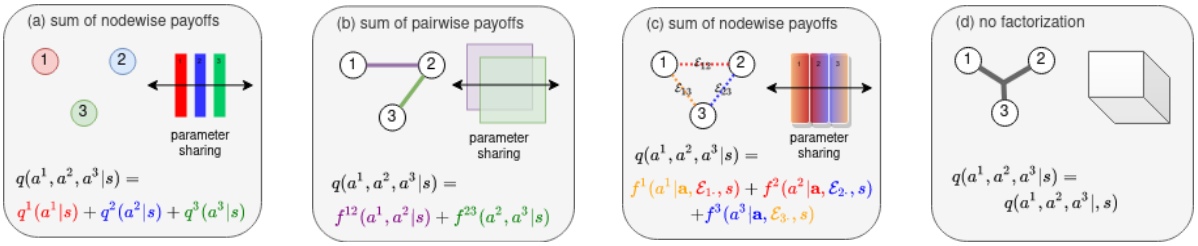
3.2 Preliminaries

Dec-POMDP We examine decentralised partially observable markov decision process [53] $\langle S, U, P, r, Z, O, n, \gamma \rangle$, with $s_t \in S$ being the environments state, and t is the

corresponding timestep, with $t = 1,2,\ldots$. At each time step every agent denoted by $a \in A \equiv \{1,\ldots,n\}$, selects $u \in U$ which is an action, with $\mathbf{u} \in \mathbf{U} \equiv U^n$ being defined as the joint action space. Additionally, we define $s' \in S$ to be the next state. This is drawn according to $P(\cdot|s,\mathbf{u})$ which is the state transition function. We are interested in cooperative RL where all agents share the same reward, so the reward for the whole environment is $r(s,\mathbf{u})$, the discount factor $\gamma \in (0,1)$. In these, the partial observable setting is considered. This is denoted to be $z \in Z$ which is generated by $O(s,i) : S \times A \rightarrow Z$. Furthermore, for a particular agent a, we define τ to be $\tau^a \in T \equiv (Z \times U)^*$, which is agent a's action observation history; on which it conditions its stochastic policy $\pi^i(u^i|\tau^i) : T \times U \rightarrow [0,1]$. Where π is the joint policy and is the Q function or the *action-value function*: $Q^\pi(s_t,\mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty},\mathbf{u}_{t+1:\infty}}[R_t|s_t,\mathbf{u}_t]$, where $R_t = \sum_{k=0}^{\infty}\gamma^k r_{t+k}$ being the total *discounted reward*. The goal of the problem is to find the optimal action value function Q^*

Centralised Training Decentralised Execution (CTDE) In this chapter, our models follow the centralised training with decentralised execution frame work. That is, training is centralised, and execution is decentralised. This framework operates by providing all state information, including global information during the centralised training phase. However each agent during execution or evaluation, can only leverage its own action-observation history.

Related Work Prior work on the decentralised agent execution include Independent Q-learning (IQL) [67], where the algorithm models each agent as an independent Q-learner. This creates challenges as a particular agent's viewpoint becomes non-stationary relative to all other agents as they also update their respective policies. To resolve this some centralised entity is used [54] [40], which typically employs some form of parameter sharing between agents. Value Decomposition Networks (VDN) [64], perform Q-learning with a value function in a centralised manner by taking the sum of each



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

Figure 3.1: Examples of value factorisation are shown in (a, b, c, d), examples of coordination graph factorisation shown in items (b, c, d): (a) sum of independent Q state action functions as used in VDN, QMIX and MAVEN; (b) sum of pairwise payoffs as used in DCG; (c) sum of node-wise payoffs based on dynamically generated coordination graph (our approach); (d) single hyper-edge with no factorisation as used in QTRAN. The factorisation indicates how the aggregated Q-value function q combines each individual parameterisation for agent i indicated by a^i.

agent's individual state value function. QMIX [57] extends VDN through employing a mixing network for the agent utilities with monotonic constraints, which allows different states to yield different mixtures so that the centralised training step can maximise the value functions in an independent manner. MAVEN [47] is another decentralised approach which builds on top of QMIX by introducing a latent space that enables hierarchical control which in turn allows for committed, temporally extended exploration. These three methods condition on the state action function of each agent and their respective historical observations, i.e. the previous actions and observations share the parameters as shown in Figure 3.1(a). DCG [5] is another decentralised agent execution approach that uses the addition of one extra linear layer which is used to approximate a coordination graphs (CG) *pairwise payoffs*. For DCG, parameter sharing is employed to approximate *all* payoff functions in the same neural network as shown in 3.1(b). QTRAN [63] is another approach which attempts to calculate the central Q-learning through applying linear constraints between agent's state action functions (or utility) and joint action values, which removes structural constraints of factorisable MARL tasks.

Individual-Global-Max (IGM) Condition and Factorisable Tasks

For CTDE methods, we define the individual agent utilities (or state action functions) by $Q^i, i \in A$. A key motivating approach for these algorithms is the ability to be *decentralised*. One approach to achieving this is the "Independent Global Max" condition [63] which states $\exists Q^i$, where for all $s, \mathbf{u}$:

$$\text{argmax}_{\mathbf{u}} Q^*(s,\mathbf{u}) = (\text{argmax}_{u^1} Q^1(\tau^1,u^1),\ldots,\text{argmax}_{u^n} q^n(\tau^n,u^n))$$

VDN and QMIX demonstrate two sufficient methods for factorising Q^* for IGM can be accomplished through additivity and monotonicity respectively. QTRAN has demonstrated that any joint action-value function $Q^*(s,\mathbf{u})$ satisfies IGM condition if

$$\hat{Q}(s,\mathbf{u}) - Q^*(s,\mathbf{u}) + \hat{V}(s) = \begin{cases} 0, & \text{if } \mathbf{u} = \bar{\mathbf{u}} \\ \geq 0, & \text{if } \mathbf{u} \neq \bar{\mathbf{u}} \end{cases} \tag{3.1}$$

where $\bar{u}^i$ is the optimal local action $\text{argmax}_{u^i} Q^i(\tau^i,u^i)$ and $\bar{\mathbf{u}} = \{\bar{u}^1,\ldots,\bar{u}^n\}$, $\hat{Q}(s,\mathbf{u} = \sum_{i=1}^n Q^i(\tau^i,u^i)$ [1], $\hat{V}$ is the state-value network defined as $\hat{V}(s) = \max_{\mathbf{u}} Q^*(s,\mathbf{u}) - \sum_{i=1}^n Q^i(\tau^i,\bar{u}^i)$. QTRAN uses a similar approach to *Deep Q-Networks* DQNs [48], which uses a *replay buffer* that stores the experiences in the tuple format $\langle s,u,r,s'\rangle$, where s' is the state observed after taking action u in state s with reward r. The parameters of the action-value function, θ is trained by using the replay buffer through minibatch sampling and minimising the following objective:

$$\begin{aligned} \mathcal{L}(\theta) :=& (Q^*(s,\mathbf{u}) - y^{\text{DQN}})^2 + \left(\hat{Q}(s,\mathbf{u}) - \perp Q^*(s,\bar{\mathbf{u}}) + \hat{V}(s)\right)^2 \\ &+ \left(\min\left(\hat{V}(s,\mathbf{u}) - \perp Q^*(s,\mathbf{u}) + \hat{V}(s), 0\right)\right)^2 \end{aligned} \tag{3.2}$$

where $y^{\text{DQN}} = r + \gamma \max_{u'} Q^*(s',u';\theta^-)$, and we let the parameter of a *target network* be defined as θ^-, which is held constant and the parameter is cloned from the θ are

[1] noting that s is the global state, and τ^i is the partial observable state for the agent i

regular predetermined intervals, and $\perp$ is the 'detach' operator which stops the gradient flow through to Q^*, which ensures Q^* is fixed.

3.2.1 Coordination Graph

A coordination graph represents the joint payoff function between agents which is aggregated to compute the joint value function in the MARL setting. In a coordination graph the vertices represent the agents whilst the edges represent the relationship between the agents of the joint action space.

In this framework, each agent i has an associated *payoff* $\mathcal{F}^i$. An agent's local utility may be influenced by other agent's actions and state, if we assume that the payoffs are independent utilities then $\mathcal{F}^i \equiv Q^i$ as shown in Figure 3.1(a) for QMIX, VDN and MAVEN models. We define $Scope(\mathcal{F}^i) \subset S \cup \mathbf{U}$ to be set of state variables and action variables that influence an agent's local utility Q^i. We further define the relevant agent decision variables to be $Relevant(\mathcal{F}^i) = \{a^j \in A | a^j \in Scope(\mathcal{F}^i)\}$. Intuitively, a coordination graph connects agents whose local utility functions interact with each other. Then extending the definition of coordination graph to the undirected counterpart proposed by Guestrin et al. [23]:

Definition 3.1. A coordination graph for a set of agents with local payoff $\{\mathcal{F}^i, \ldots, \mathcal{F}^n\}$ is an undirected graph whose nodes are $\{a^1, \ldots, a^n\}$ which contains an edge $\{a^i, a^j\}$ if and only if $a^i \in Relevant(\mathcal{F}^j)$ or $a^j \in Relevant(\mathcal{F}^i)$.

Coordination graphs depend only on state and thus every state can have its own unique CG. DCG constructs the coordination graph through consideration pairwise payoffs Figure 3.1(b), whilst QTRAN constructs a coordination graph through a single hyper-edge with no factorisation Figure 3.1(d). Our approach as shown in Figure 3.1(c), where the nodewise (or agent) payoff is constructed through consideration of the whole

state space and all agent actions. Then the total utility will depend on the setup of the respective coordination graph, as shown in Figure 3.1(b, c, d).

The application of the Heaviside step function discretises $Prob\big(a^i \in Relevant(\mathcal{F}^j)\big)$ according to threshold determined by b_{state}, it is equivalent to performing

$$\widetilde{A}_{\text{state}} = \begin{cases} 1, & \text{if } \sigma_{\text{sigmoid}}(F\mathcal{V}) \geq b_{\text{state}} \\ 0, & \text{if } \sigma_{\text{sigmoid}}(F\mathcal{V}) < b_{\text{state}} \end{cases}$$

3.2.2 Graph Neural Networks

In applying graph neural networks to the centralised training phase, in a manner which is amenable to factorisation, we need to ensure that all operations are permutation invariant to enable generalisation when learning arbitrary relationships between agents. In our thesis, we represent the undirected graph (as defined by Definition 3.1) G as $\langle A, F \rangle$, where $A \in \{0,1\}^{n \times n}$ is a symmetric adjacency matrix, and $F \in \mathbb{R}^{n \times d}$ is the node feature matrix assuming each node has d features.

Graph Convolution Network. A graph convolution network [38] (GCN) is a network which takes as input the feature matrix and the adjacency information in order to summarise the attributes of each node and output a node-level feature matrix. This operates in a similar way to how convolution operations are used where kernels are convolved across local regions of the input to produce feature maps. GCNs use layer-wise forward-propagation operation defined by $X_{(\ell+1)} = \sigma(\hat{A}X_{(\ell)}W_{(\ell)}) \in \mathbb{R}^{n \times d_{\ell+1}}$. Where $X_{(\ell)} \in \mathbb{R}^{n \times d_\ell}$ represents the feature matrix of layer ℓ with d_ℓ features, where the feature matrix at layer 0 is initialized by $X_{(0)} = F$, and where $\hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}, D_{ii} = \sum_j (A + I)_{ij}$. We denote the trainable weight matrix as $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ which applies a linear transformation to the feature vectors which is a weight matrix, and σ is the activation function. GCN is naturally permutation invariant and can be used to addressed the permutation invariance challenge in CG.

Differentiable Pooling. Whilst GCN is used for a *node level* prediction, the challenge involved generating a *graph level* prediction for the total payoff. One naive approach, is to approximate the total payoff by taking a sum, i.e. $\sum_i Q^i$ [24]. Differentiable Pooling (DiffPool) [73] allow for graph coarsening through a learnable pooling strategy, which reduces the size of the graph while retaining its essential attributes. DiffPool is a module which pools nodes given a (learned) assignment matrix which uses a layer-wise forward-propagation operation. Let the number of nodes at layer ℓ be n_ℓ, then DiffPool generates a new graph $\langle A, Z\rangle$ at layer $(\ell+1)$, defined by $A_{(\ell+1)} = V_{(\ell)}^\top A_{(\ell)} V_{(\ell)} \in \mathbb{R}^{n_{\ell+1}\times n_{\ell+1}}$ and $Z_{(\ell+1)} = V_{(\ell)}^\top X_{(\ell+1)} \in \mathbb{R}^{n_{\ell+1}\times d_\ell}$. Where $V_{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ is the learned cluster assignment matrix at layer ℓ. Then $\langle A_{(\ell+1)}, Z_{(\ell+1)}\rangle$ fully defines the new coarsened graph with $n_{\ell+1}$ nodes. $A_{(i)}$ is a real matrix, which represents the strength of fully connected edge-weighted graph. To enforce $V_{(\ell)}$ to be a proper assignment matrix, softmax function can be applied in a row-wise fashion to ensure that the assignments generated are probabilistic in nature. The learned cluster assignment matrix can be generated using a GCN layer, $V_{(\ell)} = \bar{\sigma}(AX_{(\ell)}\bar{W}_{(\ell)})$ where the learnable weight matrix $\bar{W}_{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$, and $\bar{\sigma}$ is the softmax activation function (i.e. the assignments are probabilistic). This approach has been demonstrated to be permutation invariant [73].

3.3 Proposed Method

The key idea behind QCGraph is to generate dynamic coordination graphs as part of the mixing network in the centralised training phase of the CTDE problem. In this section we explore how dynamic adjacency matrices can be generated and demonstrate how it is applied to cooperative Dec-POMDP problems.

3.3.1 Dynamic Adjacency Matrix Generation

In the previous section, both GCN and DiffPool assume a provided adjacency matrix as part of the input to the graph neural network. In the setting of coordination graphs, the adjacency matrix is defined if and only if the agents are *relevant* to each other. In this instance, we aim to have a different and dynamic adjacency matrix depending on how agents and state are reflected in the a reinforcement learning environment.

Adjacency matrix generation

We denote the learned node assignment matrix for the feature matrix as $\mathcal{V} \in \mathbb{R}^{d \times n}$ for feature matrix $F \in \mathbb{R}^{n \times d}$. Each column of $\mathcal{V}$ represents one of the n agent nodes. With a given assignment matrix, the generate adjacency matrix is defined as $A = \sigma(F\mathcal{V}) \in \mathbb{R}^{n \times n}$. Where σ is chosen to be a suitable activation function. Intuitively, $\mathcal{V}$ provides a soft assignment based on the feature matrix. The values of the adjacency matrix denotes the pairwise connectivity strength between all nodes. Then for the node pairs $\{a^i, a^j\}$, $\widetilde{A}$ will denote the measure of *relevance*. If the choice of σ is the sigmoid function, then elements i, j in $\widetilde{A}$ will denote $Prob\big(a^i \in Relevant(\mathcal{F}^j)\big)$. This output can be used as-is as input to further graph neural network layers.

Addressing Permutation invariance

For the adjacency matrix to be useful for graph classification, the generation of the adjacency matrix should be invariant under node permutations. To address this, in our construction of the dynamic adjacency matrix, the node feature matrix should be pre-sorted before computing the adjacency matrix. This can be approached as a variable to k-max pooling used in Graph U-Nets [21], where the node feature matrix is projected to 1D through a trainable projection vector $\mathbf{p}$ given by $y^i = F^i\mathbf{p}/\|\mathbf{p}\|$. Then each node can be sorted according to the scalar projection value y^i.

Adjacency Matrix Generation with Hard Assignments

Algorithm 3.1 Adjacency Matrix Generation with Hard Assignments

Input: F, S, with learnable weights $\mathbf{p}_{\text{node}}, \mathbf{p}_{\text{state}}, \mathcal{V}$

1: $F \leftarrow \text{nodesort}(F, \mathbf{p}_{\text{agent}})$ ▷ Sort nodes based on projection derived from node feature matrix as in Section 3.3.1

2: $y_{\text{state}} \leftarrow S\mathbf{p}_{\text{state}} \in \mathbb{R}$ ▷ The state specific bias through calculating projection derived from state S

3: $\widetilde{A}_{\text{agent}} \leftarrow \sigma_{\text{gumbel}}(F\mathcal{V}), \widetilde{A}_{\text{state}} \leftarrow \sigma_{\text{step}}(F\mathcal{V} + y_{\text{state}})$ ▷ Compute adjacency matrices with and without bias as in sec. 3.3.1

4: $\widetilde{A} \leftarrow \widetilde{A}_{\text{agent}} + \widetilde{A}_{\text{state}}$

5: $\widetilde{A} = \max(\widetilde{A} + \widetilde{A}^{\top}, 1)$ ▷ Ensure symmetry in undirected coordination graph, with an element-wise max function

Return: $\widetilde{A}$

With all these components in place, we can construct our approach to generate hard assignments specifically for coordination graph problem. Hard assignments enforce a binary, on/off relationship between agents. To ensure that hard assignments are generated, we use the gumbel-softmax trick [34, 46] and the Heaviside step function which are denoted by σ_{gumbel}, σ_{step} respectively. As we presume our coordination graph is undirected, symmetry is achieved through updated the adjacency matrix $\widetilde{A} = \max(\widetilde{A} + \widetilde{A}^{\top}, 1)$. This approach is given in pseudo-code form in Algorithm 3.1.

Recall in Section 3.2.1, we define an agent to be $Relevant$ to payoff for agent i is defined by $Relevant(\mathcal{F}^i) = \{a^i \in A | a^i \in Scope(\mathcal{F}^i) \subset S \cup \mathbf{U}\}$. This suggests the importance of including global state information to determine the CG from one step in our environment to another. An example of this is if there is an immediate threat outside a set of agent's partially observable state, but can be observed from a global level.

3.3.2 QCGraph

We introduce the Q-value Coordination Graph (QCGraph), which trains the payoff and value functions represented by coordination graphs. This approach aims to unify other CG approaches (QTRAN and DCG) in an overall framework, as well as remove constraints on having fixed pre-determined coordination structures. Figure 3.2 shows the neural network architecture used in QCGraph.

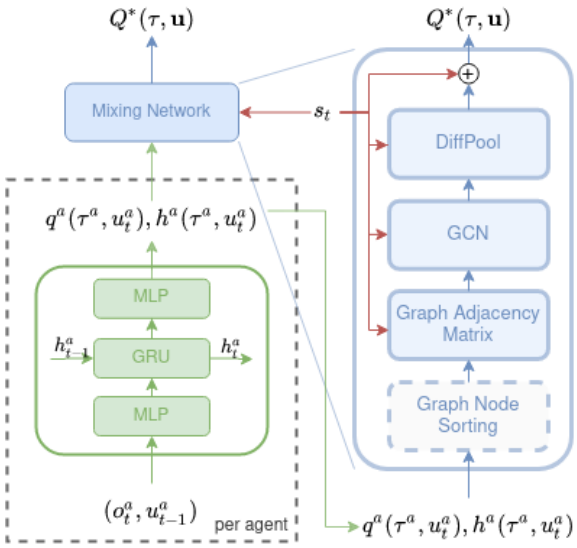


Figure 3.2: QCGraph network structure. In blue is the mixing network, and in green is the agent network structure. On the left is the overall network structure, on the right is the mixing network.

The agent network (shown in green, or bottom-left side of Figure 3.2) describes the agent utility network. Each agent's network is its individual utility or state action Q function $Q^i(\tau^i, u^i)$. At every point of time, every agent observes and uses as input its local state for a neural network constructed using a GRU recurrent layer followed by dense linear layer with $|U|$ outputs. This setup is the same one used in QMIX, MAVEN, DCG [5, 47, 57]. The output of the utility network is both the utility value itself, Q^i and also the utility network's hidden feature layer. This approach of emitting both the utility value and recurrent network's hidden state, h^i to the mixing network for centralised learning is used in both QTRAN and DCG [5, 63].

The mixing network (shown in blue, or right side of Figure 3.2). The first "module" is the graph node sorting operation. For each agent i, as Q^i can be interpreted as a 1D projection of the hidden state h^i, this can be used to pre-sort the node feature matrix as required for permutation invariance as part of the approach outlined by Equation 3.3.1. The next hypernetwork generates the graph adjacency matrix as outlined by Algorithm 3.1. The learnable parameters consists of a single linear layer and the state bias, as the

node feature matrix project (step 1. in Alg 3.1) is subsumed in the first module of the mixing network. The next two modules are the GCN and DiffPool modules explained in Section 3.2.2, which are constructed with one layer each (that is, the GCN only consists of one forward layer, and DiffPool consists of one forward layer to pool all nodes). The final state bias is constructed through a two layer hypernetwork with ReLU non-linearity, in the same manner used by QMIX. Through the use of graph neural network techniques in its setup, QCGraph generally would have comparable number of parameters or in most cases less than QMIX[2].

The agents and mixing network leverage training through centralised mechanism, and each agent uses its own utility function Q^i to take an action during decentralised execution. QCGraph is trained with the objective to minimise the loss as defined by QTRAN [63] as described by Equation 3.2, which ensures that QCGraph satisfies IGM condition.

In general CG approaches are used to avoid *relative overgeneralisation* when agents conduct exploration. DCG and QTRAN both enable the estimation of value in the overall joint action space. This approach is useful in certain environments where cooperation is important. For example, in the predator-prey environment, the predators may need to cooperate together in order to capture large or dangerous prey. From each agent's viewpoint, the reward signal will depend on other agent's actions. In models which do not take into account joint actions, (QMIX, IQl. MAVEN) it is not possible to learn the optimal policy. Furthermore QCGraph has the ability to dynamically adapt the level of coordination graph over various states which becomes important in wargame scenarios where the level of appropriate cooperation may differ according to the formation or number of agents available at a particular point in time.

[2]In QMIX, the state bias is projected to every agent in the hypernetwork. This is replaced with graph neural network modules instead.

3.4 Experiments

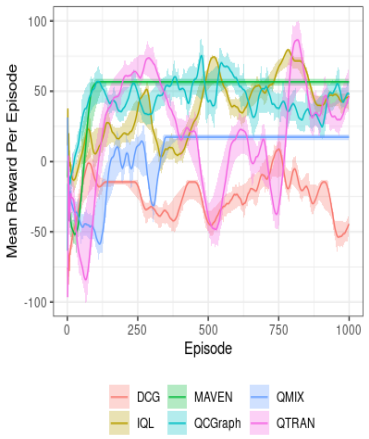
We now empirically evaluate QCGraph over several MARL benchmarks including microRTS, Modified Predator-Prey, two step and PettingZoo environments. We analyse the overal results and the representation ability of QCGraph.

3.4.1 microRTS

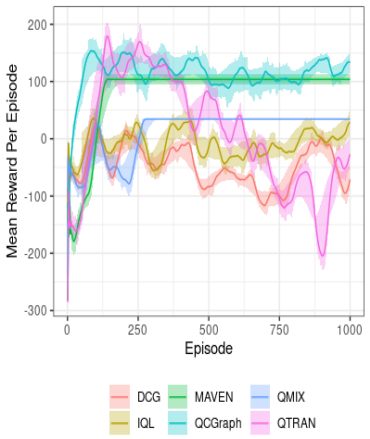
A challenging benchmark for reinforcement learning research is real-time strategy (RTS) games. Our experiments leverage an AI research focused environment called microRTS [55]. We place particular emphasis on the *decentralised micromanagement* problem in microRTS, where every agent operates an army character. The scenarios we consider are where two opposing groups of units are placed on opposing sides of the map with the aim to annihilate all opposing forces. The initial placements of units within the groups varies across episodes. For our opponent we use the winning heuristic AI agent in the 2019 microRTS competition[3]. We compare our results on sets of maps in a homogenous setting involving 4 heavy (4h), 8 heavy (8h) and 16 heavy (16h). We further compare our results using heterogeneous settings with a mixture of units involving 2 light, 2 range and 2 heavy (l2r2h2) and setups which examine different learned behaviour, involving the RL agent controlling 3 ranged units against 8 worker units and RL agent controlling 5 heavy units against 6 heavy units. The environment setup is similar to previous works which used StarCraft environments [15, 57].

First we consider the homogeneous environments (4h, 8h, 16h); these microRTS environments as shown in Figure 3.3(a, b, c). When there are a lower number of agents, IQL performs well relative to the other models. However as the number of agents increase, the performance difference between IQL and most notably QMIX begins to

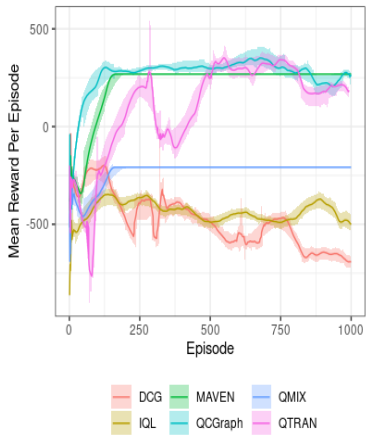
[3]Competition was hosted at IEEE CoG 2019. Results and source code of the AI bots can be found here: `https://sites.google.com/site/micrortsaicompetition/competition-results/2019-cog-results`



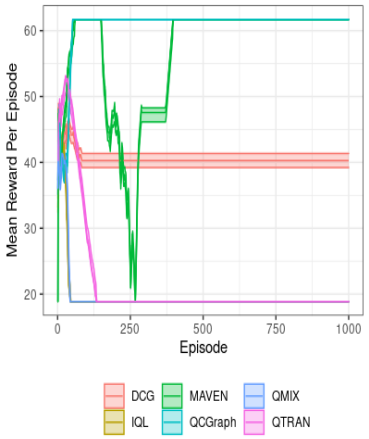
(a) 4 Heavy (Our Agent) vs 4 Heavy (Heuristic AI)



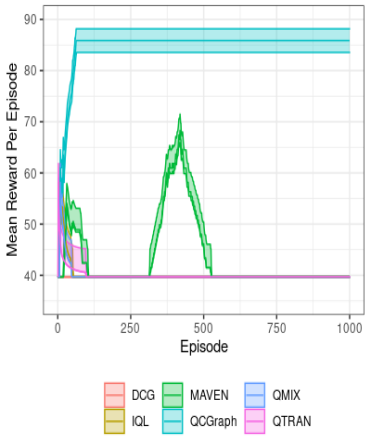
(b) 8 Heavy (Our Agent) vs 8 Heavy (Heuristic AI)



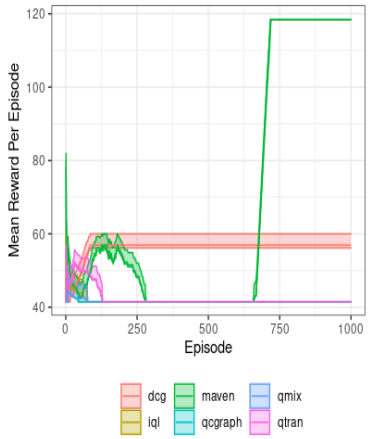
(c) 16 Heavy (Our Agent) vs 16 Heavy (Heuristic AI)



(d) 3 Range (Our Agent) vs 8 Workers (Heuristic AI)



(e) 2 Light, 2 Range, 2 Heavy (Our Agent) vs 2 Light, 2 Range, 2 Heavy (Heuristic AI)



(f) 5 Heavy (Our Agent) vs 6 Heavy (Heuristic AI)

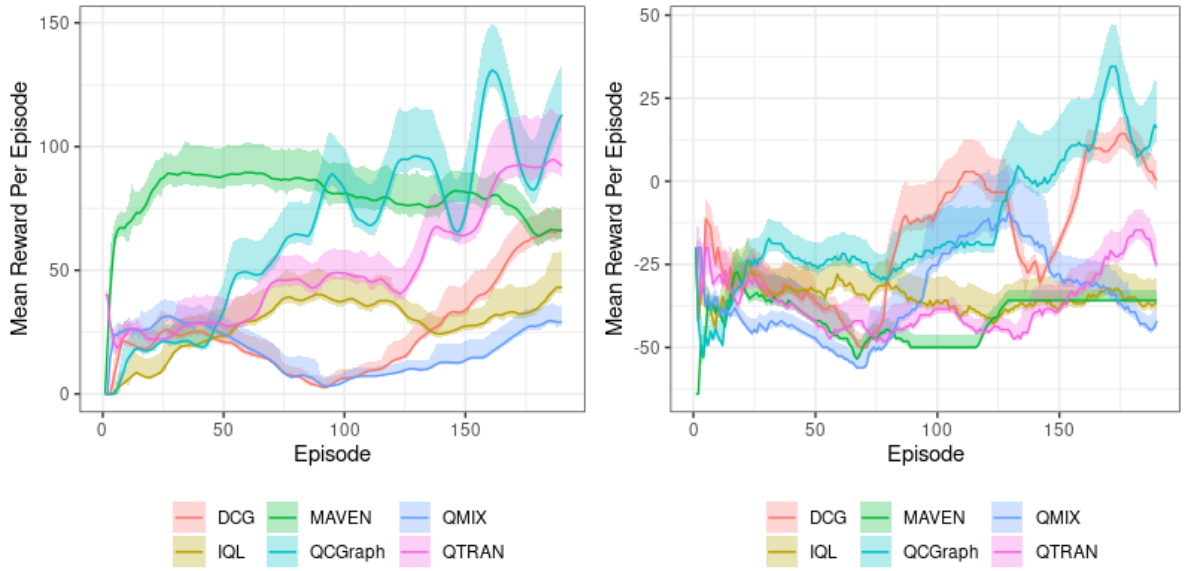
Figure 3.3: Performance of various algorithms on three microRTS maps over a variety of unit combinations. We compare the performance of our agent against the heuristic approach.

show. The flat line rewards yielded by QMIX, also demonstrate QMIX suffering from relative overgeneralisation in the microRTS environment. In contrast, in our approach, QCGraph and MAVEN continue to perform comparably with IQL in the low number agent scenario, and alongside QTRAN outperforms other approaches in the larger agent scenarios where stronger coordination is beneficial. Unlike QTRAN, QCGraph performs well in both low and high number of agents, and with improved data efficiency.

Second we consider the heterogeneous environments (r3-w8, l2r2h2, h5-h6); these microRTS environments are shown in Figure 3.3(d, e, f). As these environments are not mirror match-ups, the MARL agents are required to find novel approaches to overcoming their opponents; making use of the strengths and weaknesses of particular matchups or traits of the units. From the r3-w8 and h5-h6 variations, we can see that DCG finds a local optimal solution to coordinate the agents, but is unable to find the optimal solution. This may indicate the limitations of coordination when considering pair-wise approaches. In comparison MAVEN is able to find the best approach in both of these scenarios. Although, our approach, QCGraph does not manage to find the local or best approach in the h5-h6 scenario, we manage to find the best approaches for the mixed squad scenario and in r3-w8, suggesting QCGraph out-performs other approaches when working with longer term planning with range units which require "hit and run" tactics to perform well in scenarios where the head-on approach is inferior.

3.4.2 Modified Predator-Prey

We also explore a modified cooperative predator-prey environment (MPP) based on task presented from QTRAN and DCG [5, 63] which addresses the *relative overgenerlisation* problem. MPP is built over the microRTS game-engine where the attack action is replaced with a `capture` action to keep the prey in place, rather than damaging the prey. The prey moves randomly when it is not being held, and rewards are presented only at



(a) Modified Predator-Prey without negative rewards

(b) Modified Predator-Prey with negative rewards

Figure 3.4: Performance of various algorithms on Modified Predator-Prey.

the end. If a prey has two or more agents surrounding it a positive reward is presented (i.e. greater than 0). If a prey has one or no agent surrounding it, a non-positive reward is presented (i.e. 0 or less).

As shown in Figure 3.4 QTRAN and QCGraph perform the strongest for without negative reward penalty (the 'w/o neg') task, with QCGraph being more data efficient, whilst QCGraph and DCG perform the strongest (with QTRAN in third place) for with negative reward penalty (the 'w/ neg' task). This demonstrates the importance of CG in cooperative settings. Although MAVEN performs strongly in the 'w/o neg' MPP, it fails to converge to the optimal solution in 'w/ neg' task. Overall QCGraph appears to be the superior approach in both scenarios.

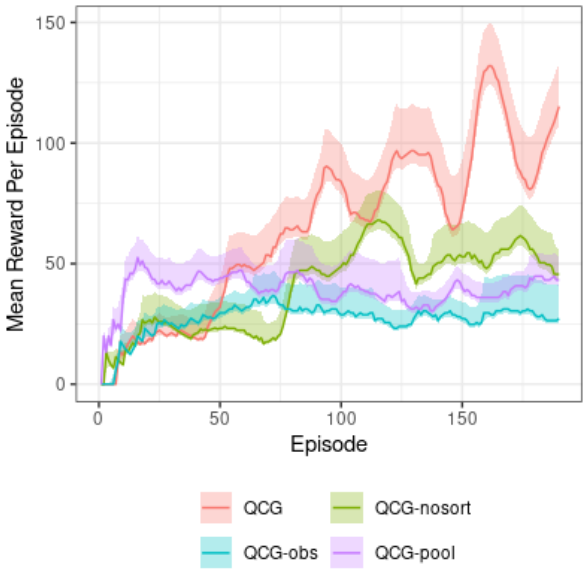
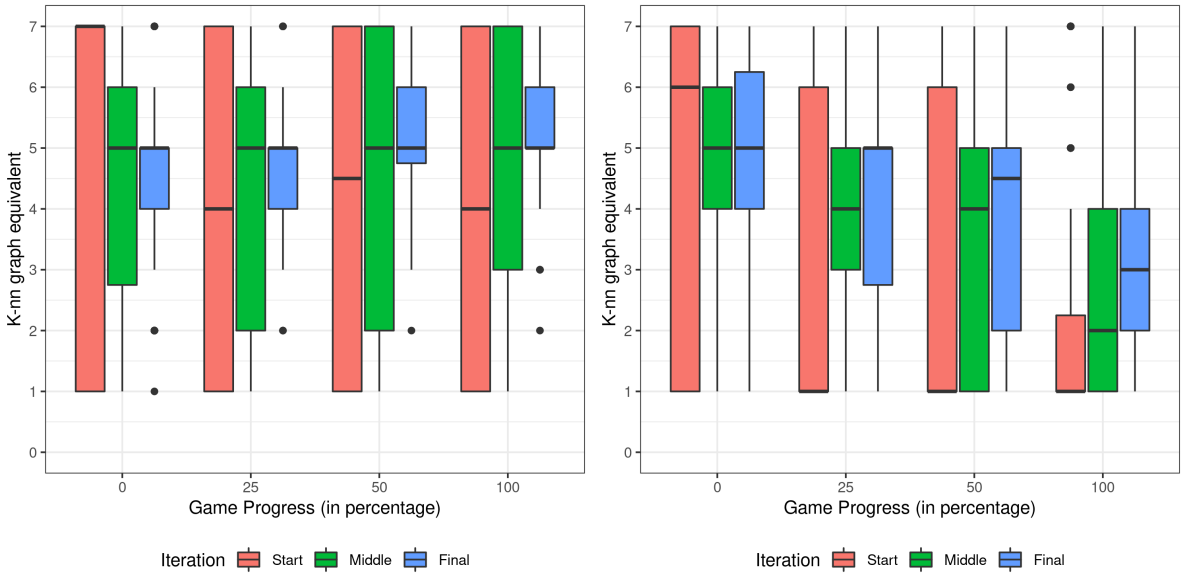


Figure 3.5: Performance of Ablations in the Modified Predator-Prey game



(a) k-nearest neighbour representation in Mod-
ified Predator-Prey where there is no negative
reward.

(b) k-nearest neighbour representation in Mod-
ified Predator-Prey where there is negative re-
ward.

Figure 3.6: Representability of Predator-Prey Games.

3.4.3 Two Step

We also use the two-step benchmark environment to demonstrate the representation ability that is possible in our approach. In the first instance only Agent 1 acts and chooses the matrix game for Agent 2. In the subsequent time step, both agents choose an action based on the matrix game in step 1 and receive a global reward according to the matrix game as shown in Figure 3.7.

In the two-step game results as shown in Figure 3.8, IQL suffer from relative generlisation, and in comparison all other approaches successfully identify the optimal payoff matrix.

Ablations We explore ablations in relation to the choice of network construction in QCGraph as shown in Figure 3.5(a). We denote the model "QCG-obs" to explore the choice of $\tilde{h}$, whether the local observation or hidden layer transformation, we label the model "QCG-nosort" for the removal of node sorting, and "QCG-pool" for the removal of DiffPool, which is replaced with global sum pooling instead (analogous to the scenario in [5] and [63]). We perform all our ablation experiments on the MPP 'w/o neg'. Without pre-sorting the nodes, the convergence is slower as it does not need to encode all possible node permutations. Without the graph neural network module, the agent fails to capture agent relationships. Using the local observation instead of the hidden layer may result in lower quality policies, as suggested in QTRAN [63], yields improved performance, which suggests noise or redundant information would be removed as part of the hidden layer representation. The use of DiffPool also allows for information to propagate for cooperative learning over global pooling layer. This demonstrates the importance in our mixing network choices and construction.

Representability The key differentiator within QCGraph compared with other coordination graph approaches is the ability to generate relations dynamically. We hypothesise that QCGraph allows dynamically generated graphs and can mold itself to

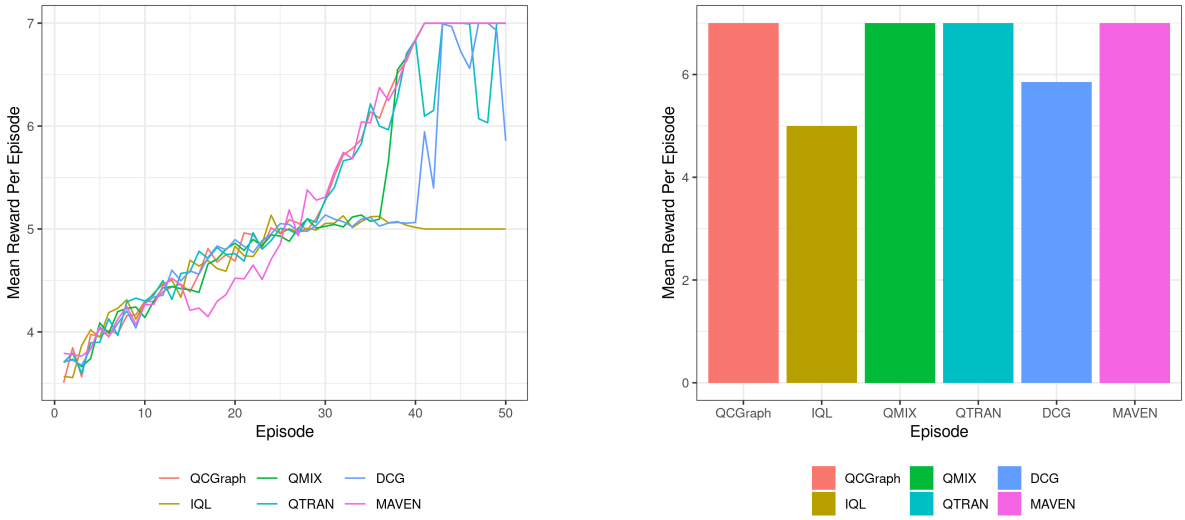
Agent 1 \ Agent 2	A	B
A	7	7
B	7	7

State 2A

Agent 1 \ Agent 2	A	B
A	0	1
B	1	8

State 2B

Figure 3.7: The two-step game's pay-off matrix after the first agent chooses an action. Choosing "A" leads to state 2A for the subsequent agent, and choosing "B" leads to state "2B".



(a) (b)

Figure 3.8: Average reward of two step matrix game (a) showing the reward in the last training iteration, and (b) showing the average reward over the training iteration.

the appropriate form, by shifting and changing the adjacency matrix structure during the mixing phase in centralised training step. To demonstrate this, we plot the *approximate* k-nearest neighbour adjacency matrix by freezing the neural network at the start, middle, and end of training for both the MPP 'w/o neg' and MPP 'w/ neg' environments. The approximate k is determined by the average number of agents deemed $Relevant$ according to Definition 3.1 per step. Each frozen model is run 100 times and the distribution of all the (hallucinated and unused) generated adjacency matrices created in the centralised training step is shown as boxplots in Figure 3.6(a, b). We observe

opposite tendencies in these two environments; when there is no penalty, QCGraph has a bias towards fully connected coordination graphs. This is reflected by QTRAN performing strongly in this environment, compared with DCG as shown in Figure 3.4(a). In comparison, in the scenario where penalties are provided, QCGraph has bias towards dynamically generating coordination graphs with lower number of agents, which explains the superior performance of DCG compared with QTRAN in 3.4(b). This demonstrates QCGraph's representational ability to adapt and build structures which transition towards more rewarding joint behaviours compared with existing CG approaches, as it exhibits more stable behaviour when the optimal solution is found.

3.4.4 PettingZoo Benchmarks

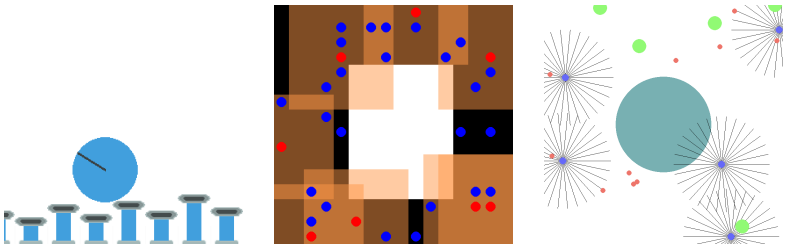


Figure 3.9: Petting Zoo Environments: pistonball, pursuit and water world.

PettingZoo offers a unified API across a multitude of MARL environments to facilitate benchmarking MARL algorithms. We access a variety of benchmark algorithms using standardised preprocessing as suggested in the original benchmarks [68]. We assess a wide variety of environments from different MARL benchmark tasks including "Butterfly" [68], "MPE" [45] and "SISL" [25] environments, with their tasks depicted in Figure 5.1. We also use variations of the environment that lower the number of agents by 25%, which have been marked by *medium* suffix in the scenarios where the default agents exceeded 4.

The "Butterfly" cooperative tasks used were *cooperative pong* and *pistonball*. In

cooperative pong the aim is to ensure the puck remains in the environment. The task terminates if the puck is out of bounds from the right or left edges of the environment.

The "MPE" cooperative tasks used were *reference* and *spread*. The *reference* task aimed to move agents towards its assigned landmark. The agent's assigned landmark is known only by the other agents. In the *spread* task, rather than having a specific landmark assigned to a specific agent, all agents must learn to cover all landmarks while avoiding collisions with one another.

The "SISL" cooperative tasks used were *pursuit* and *waterworld*. In pursuit, the agents control units (the pursuers), where the goal is to eliminate randomly moving enemies, similar to the predator prey task. An evader agent is captured when two or more pursuers surround it. In *waterworld* the pursuer agents aim to consume food while avoiding randomly moving poison.

We assess our approach, QCGraph, against QTRAN, QMIX, IQL, SEAC, LICA, and MADDPG algorithms, using the Q-learning approaches as our baseline, and comparison against the state-of-the-art MARL algorithms using actor-critic approaches.

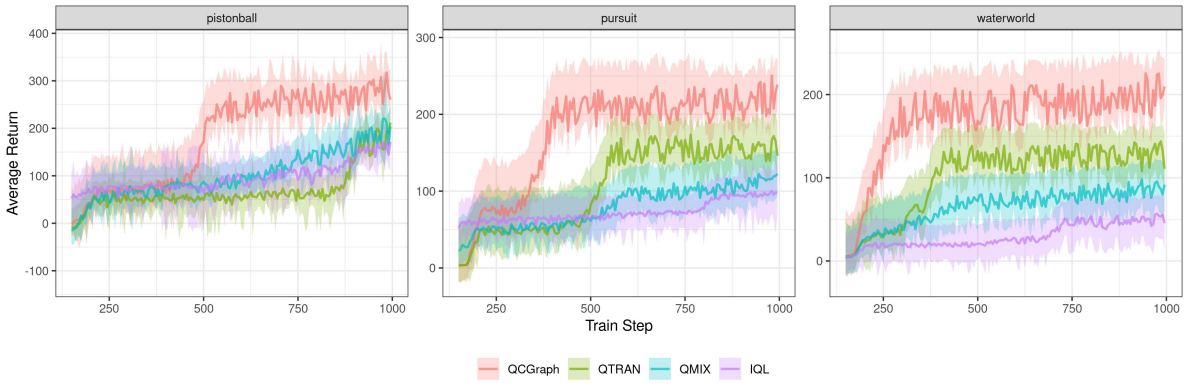


Figure 3.10: Performance of QCGraph against other Q-learning approaches over a variety of benchmark environments.

Q-learning Baseline. The performance of QCGraph against other Q-learning algorithms is shown in Figure 3.10. We observe that QCGraph yields higher returns across a range of environments. In the environments similar to predator-prey with penalties (*pur-*

	QCGraph	QMIX	IQL	SEAC	MADDPG	LICA
reference	10.64 ± 14.94	3.41 ± 18.27	-4.1 ± 16.38	*12.76 ± 17.92*	9.71 ± 16.31	**27.84 ± 19.21**
spread	*30.35 ± 28.08*	18.56 ± 27.11	12.13 ± 28.71	**42.85 ± 28.7**	19.64 ± 28.81	**43.7 ± 29.66**
pursuit	*244.2 ± 30.33*	157.13 ± 31.07	96.39 ± 32.04	168.24 ± 29.91	*179.69 ± 30.04*	**248.03 ± 29.17**
waterworld	*204.59 ± 30.24*	126.35 ± 28.87	48.78 ± 27.14	135.7 ± 27.88	156.99 ± 29.61	**206.73 ± 29.52**
pistonball	**286.02 ± 43.54**	187.43 ± 45.85	158.62 ± 39.87	*193.56 ± 49.65*	158.18 ± 30.79	172.06 ± 34.83
pong	28.91 ± 13.85	28.39 ± 8.85	22.25 ± 3.82	*46.84 ± 3.01*	**55.43 ± 5.02**	22.25 ± 2.94
pursuit medium	**169.57 ± 30.71**	29.24 ± 23.46	28.57 ± 20.91	100.84 ± 31.08	123.8 ± 29.45	*161.51 ± 29.67*
waterworld medium	**193.49 ± 30.45**	22.69 ± 21.87	32.08 ± 24.63	147.73 ± 28.21	20.39 ± 22.27	*192.47 ± 29.58*
pistonball medium	165.73 ± 55.94	58.52 ± 45.36	82.38 ± 24.95	*243.47 ± 52.34*	136.8 ± 26.95	**296.77 ± 65.96**
pong base	29.14 ± 14.87	27.42 ± 2.96	24.37 ± 8.08	**40.74 ± 13.61**	*32.81 ± 3.79*	21.2 ± 3.27
MRR default only	**0.524**	0.233	0.181	0.524	0.405	*0.512*
MRR overall	**0.561**	0.291	0.189	0.462	0.348	*0.553*

Table 3.1: Final Evaluation returns with one standard deviation. Highest and second highest returns are shown in bold and italics respectively. The mean reciprocal rank (MRR) is provided (higher is better) to allow for comparison across different MARL approaches. MRR (default only) refers to the default PettingZoo environments, whilst MRR (overall) includes the additional modified PettingZoo environments.

suit and *waterworld*), we observe QTRAN generally outperforms QMIX and IQL, which is reflected in the QTRAN results [63], but fails to yield the same level of consistency in other environments [5, 47, 76]. In this particular sets of tasks, QTRAN notably performs the worst on *pistonball* out of all Q-learning approaches. QCGraph's considerable performance improvements over Q-learning approaches confirm the effectiveness of our proposed methods.

Comparison with Actor-Critic Approaches. The performance of QCGraph against other state-of-the-art MARL approaches is shown in Table 3.1. We observe that QCGraph achieves competitive results across all tasks over a wide variety of cooperative MARL environments. Although QCGraph did not "win" more tasks than LICA algorithm, it provided more consistent results across all scenarios, demonstrated by its relatively higher mean reciprocal rank. We hypothesise this is due to the ability of QCGraph to dynamically alter its coordination structure during training to yield the ideal agent formulation in the Q-learning settings.

3.5 Conclusion

This thesis introduced QCGraph, an architecture for value factorisation over dynamically generated *coordination graphs*, which can be maximised through creating subgraphs dynamically. We evaluated our approach on a variety of environments to demonstrate the efficacy of the QCGraph algorithm. QCGraph is superior to existing coordination graph approaches experimentally, in both data efficiency and representational capability. We demonstrate how Graph Neural Networks approaches through node ordering and DiffPool can be used to resolve the agent reshuffling problem, as well as expand multi-agent reinforcement learning to a dynamic cooperative environment at both a local and global level.

Part II

Learning to Detect

Abstract

In this part of the thesis we focus on methods allowing us to *detect* changes in our experiences as we train our agents on various tasks. In chapter 4 we examine the challenge in single agent reinforcement learning where we are provided suboptimal experiences and how we can constrain our updates through detecting and modelling “good” and “bad” experiences. We evaluate our approach compared with offline reinforcement learning benchmarks. In chapter 5 we examine the multi-agent setting where we want to share experiences in DQN approach and how modifications can be made to IQL algorithms to detect and account for differences in agent experiences through importance sampling and leverage MARL benchmark environments.

4

Dual behaviour Regularisation

Contents

4.1 Introduction

In model-free reinforcement learning, the goal is to learn a policy which maximises the cumulative discounted reward for a particular environment. In most scenarios, an agent has direct access to the environment, and can perform data collection activity. This has led to development of algorithms which are able to ignore or disregard the agent's sub-optimal behaviours [1, 52]. As a consequence, greedy approaches are prevalent in reinforcement learning, leading to mode-seeking behaviour rather than approaches which model the whole space by understanding the density of returns [22, 51]. This leads to approaches which may penalise or even "forget" bad experiences and rewards, due to the inherent bias towards high rewards. This concept is known as *optimism*and can be formulated in regret settings [35].

At the same time, offline reinforcement learning has recently developed a lot of interest. This approach adds restrictions to the typical model-free reinforcement learning situation, whereby the agents may not necessarily have direct access to the environment, but instead an additional constraint is added where the model can only use experiences recorded in a static offline dataset [41, 50, 72]. This dataset of experiences may not even be optimal, and can be suboptimal or even random, where the offline RL task is for the RL algorithm to recover the policy which is emitted from the experiences (if we assume that the experiences were generated from an optimal policy) or to infer a superior policy from the sub-optimal dataset. There is particular interest in Offline RL, because in the real-world, it is typically costly to deploy or test a policy. There could also be cost or other practical considerations which prevents an agent from being able to continually update the policy in an online manner [43]. Examples of this could include autonomous driving [74], healthcare applications [49], and recommendation systems [3] which contain a large abundance of information, but the deployment of new reinforcement learning agents may only be completed after extensive testing and evaluation.

Empirically, the "greedy" nature of many RL approaches lead to poor performance in the offline setting. Even algorithms which leverage a replay buffer (DQN, DDPG, SAC) may perform poorly, which is caused by the sensitivity to the experience distribution present when applying offline reinforcement learning naively [18].

In this chapter we develop approaches to achieve the following goals: the challenge around approaches typically being "greedy" and not leveraging bad experiences, and the offline reinforcement learning setting where the experiences provided may be suboptimal with limited to no ability to access the underlying environment. Furthermore we want to ensure that any developed method can remain competitive even when used in the typical reinforcement learning scenario where the agent has full access to the environment.

To achieve this, we first want to consider the ability to leverage offline RL to provide agents the ability to learn in a structured manner. Present approaches generally require collecting experiences through interacting with the environment or presume access to such an environment. As such, many of these approaches may assume that the experiences collected are near-optimal or optimal and assume environments are consistent. We remove these constraints by introducing dual, advantage-based behaviour policy which uses counterfactual regret minimisation. Our approach is assessed in online and offline contexts, as well as scenarios where the training and evaluation environments are inconsistent. In comparing all these scenarios, the flexibility of our approach is shown. Furthermore our algorithm is comparable and even outperforms several strong baseline models in a range of continuous environments. To motivate the neural architecture choice, we perform additional ablation studies which illustrates the representative power and provide insights to how our dual behaviour regularised actor critic model has been designed, explore potential modifications, and probe the generalisability of our approach.

We leverage a behaviour regularised actor critic model to build a novel algorithm. This approach can estimate an advantage-like function which uses counterfactual regret.

From a construction standpoint it combines several ideas through Advantage-based Regret minimisation (ARM) [35] (policy variation) and Behaviour Regularised Actor Critic (BRAC) [72].

To demonstrate the efficacy of our algorithm, it is evaluated over various continuous control tasks, using the PyBullet offline reinforcement learning environment and PyBullet environments [17], and some other discrete action environments. It is shown empirically that over different RL contexts, this approach generalises better and is more robust compared with prior methods.

To tackle the challenge of *detection* our approach leverages the clipped advantage to model changes in distributions of experiences, compartmentalising them into the positive clipped advantage (or "good" experiences) and the corresponding negative clipped advantage which represents "bad" experiences.

4.2 Background

It has been confirmed empirically that using off-policy approaches such as TD3 and DDPG in offline reinforcement can struggle to learn a sensible policy, even when the offline experiences presented are derived from a single, consistent (optimal) behaviour [72]. This challenge is due to the erroneous generalisation which is derived from leveraging function approximators to estimate/approximate the Q, state-action value function. One way this has been addressed is through regularising the learned policy towards the behaviour policy explored in Behaviour Regularised Actor-Critic (BRAC) [72] and Bootstrap Error Accumulation Error (BEAR) [41]. The difference between these approaches centres around the choice of divergence metrics and choice of the regulariser between the behaviour and target policy. For example, Maximum Mean Discrepancy (MMD) is used by BEAR and Kullback-Leibler divergence is used by BRAC [41, 72].

Rather than explicitly regularising policies, other approaches would be to *implicitly*

regularise policies to support offline reinforcement learning. One approach is the advantage weighted actor-critic algorithm (AWAC), which uses implicit constraint on the actor with an off-policy critic. In this particular scenario, AWAC does not use an explicit behaviour policy, as it learns a policy that maximises the critic function via its value function, but also implicitly restricts the distribution of the policy to remain within the range of observed data during the update step of the actor network [50]. In contrast, Munchhausen Reinforcement Learning (MRL) [70], which uses implicit regularisation has been used in Q-learning by augmenting the immediate reward. This approach augments the immediate reward with the scaled log-policy which has been demonstrated to implicitly perform KL regularisation when executing the policy update. Another related approach is to perform reward decomposition as part of the off-policy training, however current approaches use this only to aide explainability and not learning [36].

4.3 Dual behaviour Regularised Reinforcement Learning

Next we provide the necessary background of our approach "Dual Behaviour Regularised Reinforcement Learning" (DBR). Then we explore the representation power of DBR and provide intuition behind how DBR operates.

4.3.1 Behaviour Regularised Actor-Critic

In this section, we formally introduce Behaviour Regularised Actor-Critic approaches including BEAR [41] and BRAC [72], which are the basis of our Dual Behaviour Regularised algorithm and are used in *offline* reinforcement learning.

First we consider the Soft Actor-Critic (SAC) framework which underpins both BEAR and BRAC. The modification which SAC makes compared with the actor-critic definition

provided above, is the addition of an entropy regulariser term. In the SAC framework, we can define the expectation over a given state which induces the value function as below:

$$V_\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}}(R_t|s_t) \tag{4.1}$$

Then to cater for the additional entropy regulariser term, we modify Equation 4.1 to:

$$V_\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}}[R_t + \alpha H(\pi)|s_t] \tag{4.2}$$

Then to extend SAC to the BRAC and BEAR framework, we replace the entropy regulariser with a choice of Divergence function D with the appropriate behavioural policy β.

$$V_\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}}\left[R_t - \alpha D(\pi, \beta)|s_t\right] \tag{4.3}$$

In this setting the behavioural policy β is generated from the static dataset of transitions $\mathcal{D}$, which means that the behavioural policy is always well-defined, even if multiple distinct policies were used to collect experiences. As a consequence, it cannot be presumed that the algorithm can directly access the behaviour policy. Instead we approximate this behaviour policy [41, 72].

$$\hat{\beta} := \underset{\hat{\pi}}{\operatorname{argmax}} \mathbb{E}_{(s,u,r,s')\sim\mathcal{D}}[\log \hat{\pi}(u|s)] \tag{4.4}$$

The differences between BRAC and BEAR lie in the choice of divergence functions. In the BRAC framework, the Kulback-Leiber divergence is used, whilst the Maximum Mean Discrepancy divergence (MMD) used in the BEAR framework additionally adds a threshold constraint $\epsilon \geq 0$ to the kernel MMD distance, which is shown below:

$$\mathbb{E}_{s\sim\mathcal{D}}[D(\pi, \beta)|s_t] < \epsilon \tag{4.5}$$

This grants the algorithm control over the level of constraint the policy can diverge away from the behaviour policy. In theory, altering ϵ enables faster or slower fine-tuning when the BEAR algorithm is allowed to collect samples of experiences [50].

4.3.2 Clipped Advantage Actor-Critic

In this section, we examine two approaches which both use clipped advantage and their intuitions to improve actor critic approaches. First self-imitation learning (SIL) [52] builds ontop of SAC. In this formulation an additional update step is performed in conjunction with the standard update procedure. This update step uses the clipped advantage to update the actor-critic model. Within SIL, the clipped advantage is shown below

$$A_t^{\text{sil}}(s,u) = \max\left(\sum_{k=t}^{\infty} \gamma^k r_k - V(s;\theta), 0\right) \tag{4.6}$$

where r_k indicates the reward at time k and γ being the discount rate.

The second approach is the advantage regret minimisation framework (ARM), where the clipped advantage is in the recursive form:

$$A_t^{\text{arm}}(s,u) = \max(A_{t-1}(s,u), 0) + (Q(s,u;\theta) - V(s;\theta)) \tag{4.7}$$

Beyond the recursive nature of ARM and SIL, the intuitive difference between the two approaches is the definition of the advantage function. In SIL the advantage is in the form that considers the cumulative discounted rewards, whereas the default approach in ARM considers advantage in the form that leverages the Q-function, though any advantage function can be used, including the formulation used in SIL.

The other difference in the two approaches is the algorithmic setup. In SIL, the authors propose an additional step in training the policy to perform an additional update,

where it functionally speaking resamples the agent's experience only taking the top experiences to retrain, whereas ARM formulation does not require an additional update step and can use the clipped advantage directly in its training regime for the policy.

One of the challenges in the ARM framework, is the potential in the policy update returning log(0). Due to this, ARM is typically used only in Q-learning due to the discrete action space. This does not occur in the SIL setup because experiences which would cause such an update would have been discarded.

4.3.3 Method

One of the main contributions of our algorithm, Dual Behaviour Regularised Actor-Critic (DBR), is to remediate the constraints of the BEAR and ARM algorithm. DBR aims to address the following deficiencies:

- Recall in the BEAR algorithm, the existence of a *fixed* hyperparameter ϵ which allows the BEAR policy to have greater flexibility when fine tuning. Can we enable a flexiblity ϵ?
- In the ARM formulation, we are restricted to Q-learning approaches spaces due to the nature of clipped advantage. Can we extend it to actor-critic approaches without encountering the log(0) issues which would occur?

To address these points, DBR splits the behavioural policy estimation into two parts, the *positive* and *negative* clipped advantage, which are defined in Equation 4.8 and 4.9 below:

$$A_t^+(s,u) = \max\left(\sum_{k=t}^{\infty} \gamma^k r_k - V(s;\theta), 0\right) \tag{4.8}$$

$$A_t^-(s,u) = \min\left(\sum_{k=t}^{\infty} \gamma^k r_k - V(s;\theta), 0\right) \tag{4.9}$$

Furthermore we split our Behavioural policies into *positive* and *negative* variations in a similar manner

$$\hat{\beta}^{+} := \underset{\hat{\pi}}{\operatorname{argmax}} \mathbb{E}_{(s,u,r,s')\sim\mathcal{D}_{+}}[\log \hat{\pi}(u,|s)] \tag{4.10}$$

$$\hat{\beta}_{-} := \underset{\hat{\pi}}{\operatorname{argmax}} \mathbb{E}_{(s,u,r,s')\sim\mathcal{D}_{-}}[\log \hat{\pi}(u,|s)] \tag{4.11}$$

$$\tag{4.12}$$

where the replay buffer subsets that revolve around the value function are defined as $\mathcal{D}_{+}, \mathcal{D}_{-}$. That is $\mathcal{D}_{+} = \{(s,u,r,s') \in \mathcal{D} \text{ s.t. } A^{+}_{t}(s,u) > 0\}$ and $\mathcal{D}_{-} = \{(s,u,r,s') \in \mathcal{D} \text{ s.t. } A^{-}_{t}(s,u) < 0\}$. The update for $\hat{\beta}^{+}$ and $\hat{\beta}^{-}$ uses the same behavioural policy and only controls the action space in which the actor is trained and penalised against.

We can apply an ARM style update to the BEAR framework which would resemble

$$V_{\pi}(s_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}} \left[R_t - \alpha D(\pi, \hat{\beta}^{+}) | s_t \right] \tag{4.13}$$

with the divergence function choice to be MMD. Then to enable the policy improvement step to be dynamic, rather than using a fixed ϵ, we instead can leverage the negative behaviour policy

$$\mathbb{E}_{s\sim\mathcal{D}}[D(\pi, \hat{\beta}^{+})|s_t] < \max(\epsilon, \mathbb{E}_{s\sim\mathcal{D}}[D(\pi, \hat{\beta}^{-})|s_t]) \tag{4.14}$$

Intuitively, as new experiences are added to the replay buffer, a desirable outcome is for the divergence between the estimated positive and negative behaviour policies, moving towards the positive behaviour policy than the negative behaviour policy, i.e. $D(\pi, \hat{\beta}^{+}) < D(\pi, \hat{\beta}^{-})$. This idea is central to our *detection* challenge, as the dual behaviour policies act as a surrogate to detect and adapt to changing distributions and experiences.

This approach avoids the issue of discarding experiences as in SIL and the policy gradient update in ARM by training $\hat{\beta}^{+}, \hat{\beta}^{-}$ concurrently. Then all experiences would

be used in the counterfactual regret minimisation framework, and achieved through surrogate objective optimisation of $\hat{\beta}^+$, instead of analysing the clipped advantage directly. Consequently this implicitly ensures the objective is as below

$$\underset{\pi}{\operatorname{argmin}} \mathbb{E}_{s\sim\mathcal{D}}[D(\pi, \beta^+|s)] \tag{4.15}$$

which means the behaviour policy β^+ is approximated by using the clipped advantage A^+.

Therefore, our approach is a variation of ARM with alterations of the surrogate objective, thereby addressing the deficiencies in the original ARM algorithm and allowing it to be used in policy gradient and actor-critic approaches.

In spite of the modifications, DBR still maintains the concentrability coefficient bound in the original BEAR algorithm which considers the dual data distributions $\mathcal{D}^+$ and $\mathcal{D}^-$ and its corresponding marginal state distribution. Define Π to be the set of policies, and $\Pi_\epsilon = \{\pi|\pi(u|s) = 0 \text{ whenever } \beta^+(u|s) < \max(\epsilon, \beta^-(u|s)\}$ to be Π when constrained by support sets (i.e. policies that reside in the positive clipped advantage, but away from negative clipped advantage regions). Then the concentrability coefficient can be defined as:

Assumption 4.1. (Concentrability) *Define ρ_0 to be the initial state distribution, and the distribution of training data $S \times U$ be defined as $\mu(s,u)$, with marginal $\mu(s)$ over S. If there exists coefficients $c(k)$ where $\pi_1, \ldots, \pi_k \in \Pi$ and $s \in S$:*

$$\rho_0 P_1^\pi P_2^\pi \ldots P_k^\pi(s) \leq c(k)\mu(s) \tag{4.16}$$

where transition operator on states induces by π_i is defined by P_i^π. Then, we can define $C(\Pi)$, the concentrability coefficient as

$$C(\Pi) := (1-\gamma)^2 \sum_{k=1}^{\infty} k\gamma^{k-1} c(k) \tag{4.17}$$

Theorem 4.1. *Define the data distribution created by behaviour policy β_i to be μ_{β_i}. Let the marginal state distribution be $\mu_{\beta_i}(s)$, $\Pi_\epsilon = \{\pi | \pi(a|s) = 0$ if $\beta_+(a|s) < \max(\epsilon, \beta_-(a|s))\}$, and the highest discounted marginal state distribution be defined as μ_{Π_ϵ}, with ρ being the distribution of the initial state which follows $\pi \in \Pi_\epsilon$ afterwards. Then $C(\Pi_\epsilon)$, the concetrability coefficient exists and is bounded:*

$$C(\Pi_\epsilon) \leq C(\beta_+) \cdot \left(1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)\right) \tag{4.18}$$

where $f(\epsilon) := \max_{\tilde{\beta} \in \{\beta_+, \beta_-\}} \min_{s \in S, \mu_{\Pi_\epsilon}(s) > 0} [\mu_{\tilde{\beta}}(s)] > 0$.

Proof. For clarity we simplify notation so Π_ϵ is referred to as Π. Then let μ_Π to be highest discounted marginal state distribution from initial starting state ρ with policy $\pi \in \Pi$. That is:

$$\pi_\Pi := \max_{\{\pi_i\}_i : \forall i, \pi_i \in \Pi} (1-\gamma) \sum_{m=1}^{\infty} m \gamma^{m-1} \phi_0 P_1^\pi \cdots P^{\pi_m} \tag{4.19}$$

By definition of Π, we have $\Pi, \forall s \in \mathcal{S}, \forall \pi \in \Pi, \pi(u|s) > 0 \Rightarrow \beta_+(u|s) > \epsilon$. The marginal state distribution of β_+ and Π can be shown to be bounded in total variation distance by $D_{TV}(\mu_{\beta_+} || \mu_\Pi) \leq \frac{\gamma}{1-\gamma}(1-\epsilon)$.

By definition of Π, which is the set of policies, $\forall s \in \mathcal{S}, \mu_\Pi(s) > 0 \Rightarrow \mu_{\beta_+} \geq f(\epsilon)$. Where $f(\epsilon) > 0$ is a constant with parameter ϵ. This observation holds irrespective of the BEAR or DBR framework. This encapsulates the maximum visitation probability of states when executing behaviour policy $\beta_+(a|s)$ or $\beta_-(a|s)$ from ρ, the initial state distribution, with $\beta_+(a|s) \geq \epsilon$ which constrains the actions which can be executed from the environment. Then in consideration of $\beta := \beta_+ \cup \beta_-$, we have the constant $f(\epsilon) \geq \beta$. When combined with the total variation diverse bound, $\max_s \| \mu_{\beta_+}(s) - \mu_\Pi(s) \| \leq \frac{\gamma}{1-\gamma}(1-\epsilon)$, we get that

$$sup_{s \in S} \frac{\mu_\Pi(s)}{\mu_{\beta_+}} \leq \frac{\gamma(1-\epsilon)}{(1-\gamma)f(\epsilon)} + 1 \tag{4.20}$$

Given, $C(\Pi) := (1-\gamma)^2 \sum_{k=1}^{\infty} k\gamma^{k-1} c(k)$ or the marginal state visitation distribution ratio when performing policy iterate with backups using the distribution-constrained operator and $\mu = \mu_{\beta_+}$ the data distribution. As a result,

$$\frac{C(\Pi_\epsilon)}{C(\beta_+)} := \sup_{s \in S} \frac{\mu_\Pi(s)}{\mu_{\beta_+}} \leq 1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon) \tag{4.21}$$

■

Our approach differs from the BEAR algorithm as it uses β_- to intentionally avoid performing poor actions. This is particularly important if there are degenerative states as part of the environments, which are non-terminal states that will create sub-optimal trajectories. An example of such an environment is shown in Figure 4.1 as part of the Connect-4 game, which demonstrates our algorithms ability to avoid critical mistakes compared with other approaches. This is shown in the learned policy behaviour output shown in Figure 4.1(b), where our approach successfully identifies the correct move in constrast to approaches which only use β_+.

If DBR uses a fixed constraint like BEAR rather than using β_-, then our model would take the form of ARM where the behaviour policy acts as a constraint enforcing clipped-advantage style update. If the form of the advantage function uses the cumulative reward, instead of the Q state action function with a separate update step, then the model would take the form of SIL instead. Rather than performing explicit regularisation, we can also augment the reward similar to Munchhausen Reinforcement Learning, which performs an implicit entropy regularisation. To regularise with respect to the behaviour policy, we can modify the reward augmentation to the form $r_t^{\text{mrl}} := r_t + \kappa \log \beta_+(u|s)$, with the scaling factor $\kappa \in [0,1]$. In our ablation experiments, we explore all of these variations.

4.4 Experimental Results

We compare the results of our DBR algorithm against other approaches. First we describe our experiment details including data collection and hyperparameters, and then we discuss the details of our results.

The experiments we conduct are both for offline reinforcement learning and online reinforcement learning. We also evaluate DBR on environments where the training environment is different (easier) to solve than the evaluation environment to re-affirm the representative power of our approach.

Our experiments were conducted by leveraging the PyBullet environments in the online setting and also with generated offline PyBullet datasets. The choice of hyperparameters for comparison networks is based on their official implementation details and is shown in Table 4.1. We use a three layer neural network in all of our experiments and attempt to keep all the neural network parameter sizes the same across all approaches with appropriate changes on an algorithm level. We compare our approach with TD3, SAC, BEAR and AWAC. For DBR specifically, we leverage the same hyper-parameter setup as BEAR. This included the same settings for Q function ensembles with two ensembles. We also conduct ablation studies around the variations of DBR which leverage self-imitation or Munchhausen RL approaches which used the original author's suggested hyperparameters. All experiments were evaluated using 1000 episode rollouts which use entirely different evaluation environments, and we show the training iteration and average reward in the respective tables and figures.

4.4.1 Data collection

To generate data for our offline experiences dataset, we leverage Soft Actor-Critic algorithm. The breakdown of our approach is shown below:

- Random data was generated through sampling based on the uniform distribution over the action space in the environment
- The medium dataset was generated by learning a policy using the Soft Actor Critic algorithm for 1 million timesteps for each environment and using the final policy to generate a dataset of experiences.
- Mixed data was created through including all experiences of the agent throughout the training of the medium agent.

4.4.1.1 Hyper-parameters

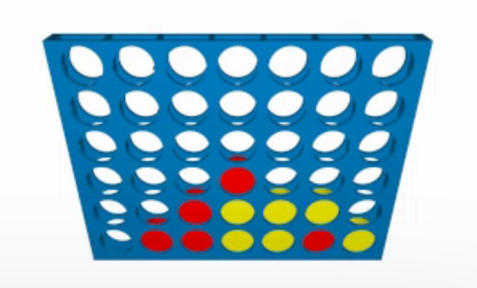
Hyperparameter	**Values**
Replay Buffer Size	1000000
Steps per Iteration	1000
Number of pretraining steps	1000
Learning Rate	0.0003
Batch Size	256
Policy Hidden Sizes	[256, 256, 256]
Policy Hidden Activation	ReLU
Target Network τ	0.005
Reward Scale	1
Discount	0.99

Table 4.1: Hyper-parameters used for RL experiments

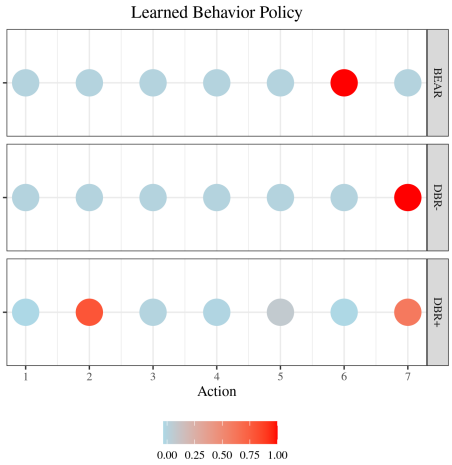
4.4.2 Connect-4

For this task, we demonstrate the representation power and the ability for DBR to generalise by having the RL agents train on an agent which acts according to a random policy, whilst evaluating on a heuristic agent which leverages Monte-Carlo Tree Search [7]. We leverage Double DQN [29] as our benchmark and modify BEAR and DBR to use discrete actions [10]. The results can be seen in Figure 4.1 and in Section 4.3.3 where it is

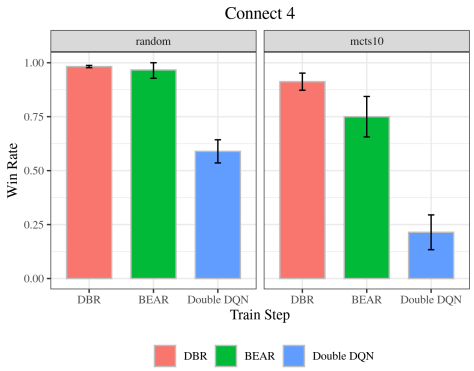
[1]From "gameplay of Connect-4", "`https://en.wikipedia.org/wiki/Connect_Four''`



(a) Connect-4: Red to play. The only move which does not lose the game is the right-most file which blocks Yellow's winning move[1]



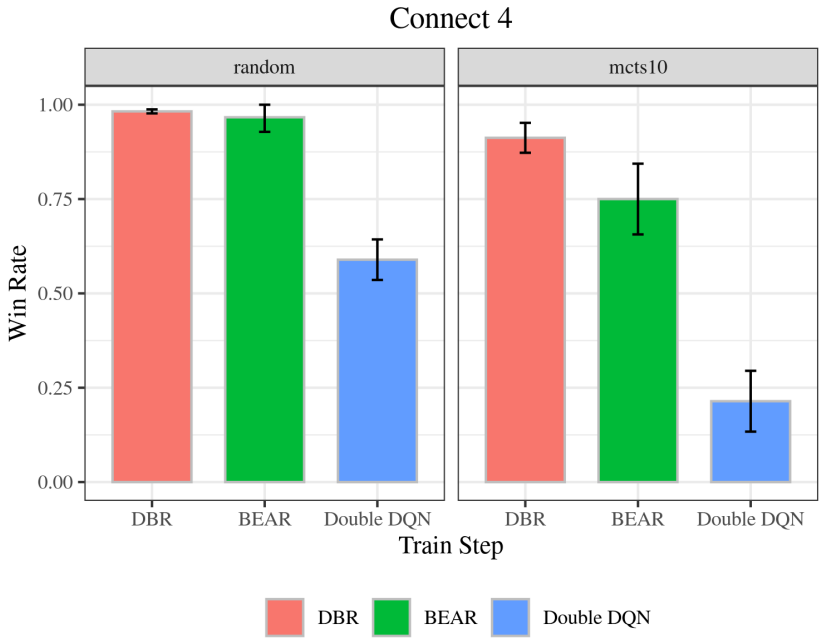
(b) Learned behaviour policy, based on the environment state depicted in (a).



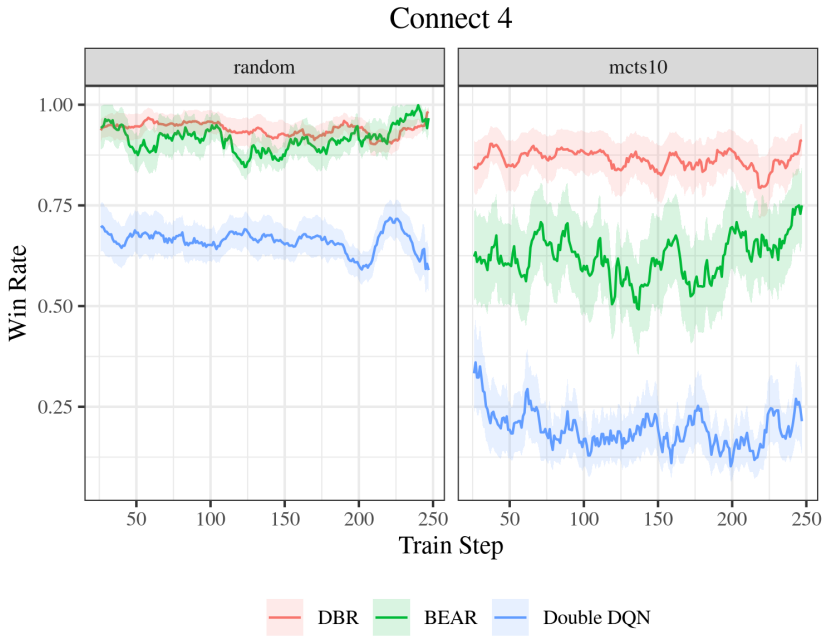
(c) Evaluation performance using an (unseen) adversarial agent (depth 10 MCTS).

Figure 4.1: Connect-4: during training, the opposing agent makes moves at random. However it is evaluated on a separate and unseen agent which acts according to a MCTS policy with various depths. All of the different approaches were trained for two million rollouts. Notice that DBR β_- has highest energy with the right move. Contrast this with only using β_+ which fails to identify the correct action. In comparison BEAR completely fails to identify the correct action.

demonstrated that DBR is able to generalise and that there is a loss in performance from the other agents as we move from an easy training environment to a more challenging evaluation environment as shown in Figure 4.2. This demonstrates that DBR may benefit from a more diverse set of experiences, even if this includes lower-quality examples to construct superior policies.



(a)

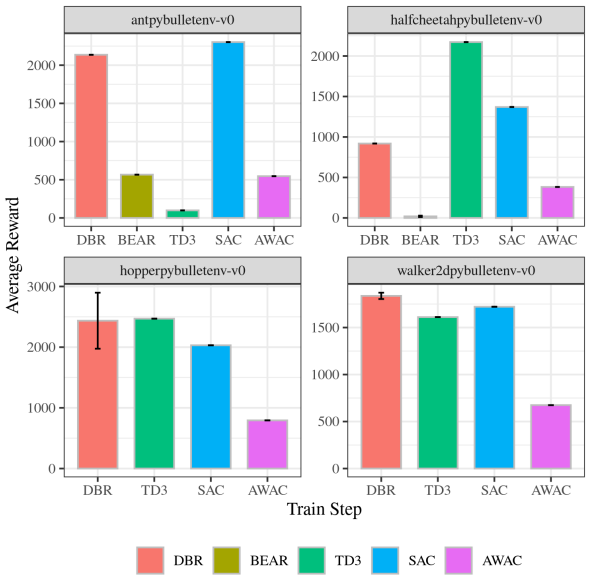


(b)

Figure 4.2: Performance over Connect-4 environment, where the agents are trained on a random agent, and evaluated on a random agent (left) and a heuristic MCTS agent with depth 10 (right). The average is provided over the last ten steps, with error bars representing one standard deviation.

4.4.3 PyBullet Environments

When training on the PyBullet environments, we allow agents access to the environment. Although algorithms like BEAR, AWAC were designed to be used in the offline or batch setting rather than the online setting, this task demonstrates how our modifications to BEAR algorithm enable DBR to dynamically control exploration. The outputs of our experiment is shown in Figure 4.3 where we observe DBR is indeed superior to other offline RL approaches (AWAC, BEAR), and comparable to the typical baselines (SAC, TD3). One key observation around the limitation of AWAC, is the necessity of building up the policy through offline data, which may prevent AWAC activating *tabula rasa*. Throughout this process, it is demonstrated that DBR is able to find the appropriate balance between offline and off-policy reinforcement learning.



(a)

Figure 4.3: PyBullet environments: trained tabula rasa. The graphs report last ten steps with the error bars being one standard deviation.

Tasks/Algorithms	DBR	DDQN	TD3	SAC	BEAR	AWAC
Ant-PyBullet (Offline - Random)	$\mathbf{13.6 \pm 5.73}$	–	11.2 ± 5.99	10.1 ± 2.4	11.1 ± 6.02	10.0 ± 4.15
Halfcheetah-PyBullet (Offline - Random)	-226 ± 359	–	180 ± 133	-192 ± 8.2	-1405 ± 42.7	$\mathbf{301 \pm 4.6}$
Hopper-PyBullet (Offline - Random)	$\mathbf{16.6 \pm 2.88}$	–	11.9 ± 0.856	16.2 ± 2.86	16.2 ± 2.93	16.1 ± 2.68
Walker2d-PyBullet (Offline - Random)	0.822 ± 0.490	–	4.69 ± 0.426	1.45 ± 0.465	$\mathbf{33.7 \pm 4.97}$	13.0 ± 1.08
Ant-PyBullet (Offline - Medium)	175 ± 2	–	$\mathbf{266 \pm 53.9}$	106 ± 2	91 ± 18.25	169 ± 2
Halfcheetah-PyBullet (Offline - Medium)	-1539 ± 8.2	–	-1716 ± 14.7	$\mathbf{-49.9 \pm 9.2}$	-1359 ± 312	-1456 ± 6.4
Hopper-PyBullet (Offline - Medium)	$\mathbf{17.3 \pm 6}$	–	15.4 ± 0.65	11.5 ± 2.6	16.1 ± 2.97	14.1 ± 6.17
Walker2d-PyBullet (Offline - Medium)	33.3 ± 4.8	–	8.03 ± 7.83	36.6 ± 6.81	$\mathbf{46.5 \pm 49.3}$	14.6 ± 19.8
Ant-PyBullet (Offline - Mixed)	292 ± 0.255	–	43.7 ± 24.9	236 ± 0.153	82.4 ± 7.32	$\mathbf{403 \pm 0.119}$
Halfcheetah-PyBullet (Offline - Mixed)	-1694 ± 9.3	–	-1675 ± 19.2	-1654 ± 8.7	-1669 ± 14.78	$\mathbf{-870 \pm 3.1}$
Hopper-PyBullet (Offline - Mixed)	12.1 ± 3.38	–	13.5 ± 0.783	18.3 ± 27.9	16.1 ± 2.88	$\mathbf{23.1 \pm 13.6}$
Walker2d-PyBullet (Offline - Mixed)	39.3 ± 7.99	–	8.74 ± 21.2	35.6 ± 5.51	$\mathbf{52.3 \pm 61.2}$	48.4 ± 15.5
Ant-pybullet-v0 (Online)	2136 ± 6.5	-	98.9 ± 3.4	$\mathbf{2302 \pm 7.5}$	567 ± 9.3	547 ± 9.2
Halfcheetah-pybullet-v0 (Online)	919 ± 9.4	–	$\mathbf{2172 \pm 4.5}$	1370 ± 3.2	19.3 ± 18	383 ± 2.8
Hopper-pybullet-v0 (Online)	2435 ± 923	–	$\mathbf{2470 \pm 9.3}$	2031 ± 7.4	33.4 ± 1.83	794 ± 6.1
Walker2d-pybullet-v0 (Online)	$\mathbf{1837 \pm 758}$	–	1611 ± 117	1721 ± 7.39	47.0 ± 16.9	6.75 ± 5.82
Connect 4 ‚Äì Random	$\mathbf{0.982 \pm 0.0158}$	0.589 ± 0.161	–	–	0.967 ± 0.116	–
Connect 4 ‚Äì MCTS 10	$\mathbf{0.912 \pm 0.0793}$	0.214 ± 0.161	–	–	0.75 ± 0.187	–

Table 4.2: Performance in benchmark tasks. We report the performance based on the evaluation environment with one standard deviation.

4.4.4 Offline Reinforcement Learning

Next we examine the performance of DBR over offline RL tasks. We divide our analysis in two parts, corresponding to the environments available: medium-quality data, and performance on random/mixed quality data.

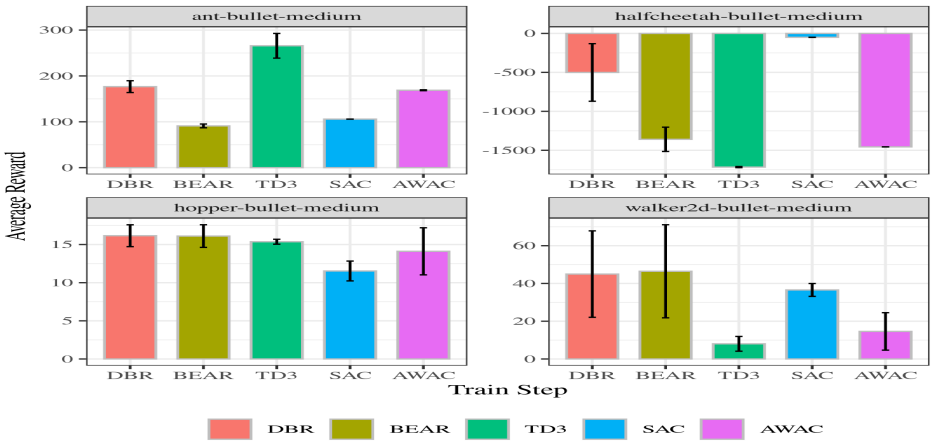
For the medium quality dataset, we observe that DBR is comparable with BEAR - having similar performance or better (Figure 4.4(a)). In the scenarios where DBR was outperformed by TD3 or SAC, DBR was the next strongest approach. This is shown in our results for environments “ant-PyBullet” and “halfcheetah-PyBullet”.

For the random and mixed datasets, we observe that DBR achieves good results compared with other approaches except for “walker2d-PyBullet-random” and “hopper-PyBullet-mixed” environments. These results shown in Figure 4.4 are in line with our expectations that DBR is a more robust algorithm, and can adapt more readily in light of suboptimal experiences in the dataset.

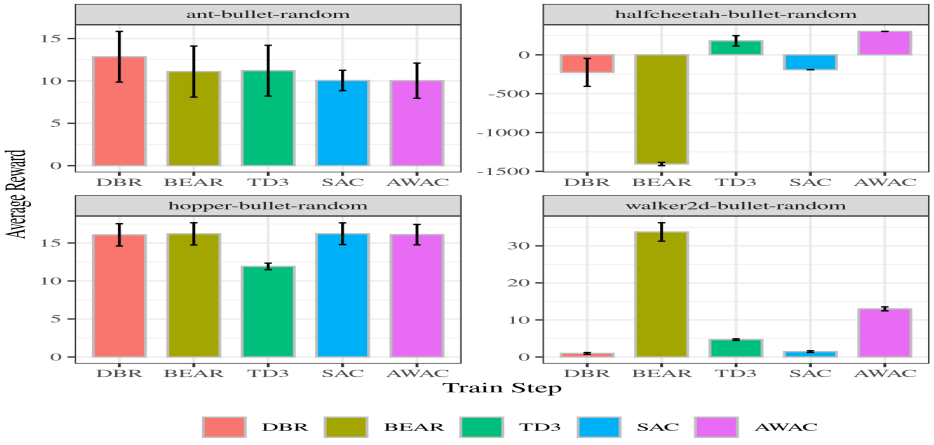
4.4.5 Ablation Studies

In our DBR formulation we have been prescriptive in our algorithmic choices. In this section we empirically explore the algorithm decisions, justify our choices over a range of environments and demonstrate the stability of DBR. We also explore the possibility of using an implicit regularisation scheme in a similar manner to MRL.

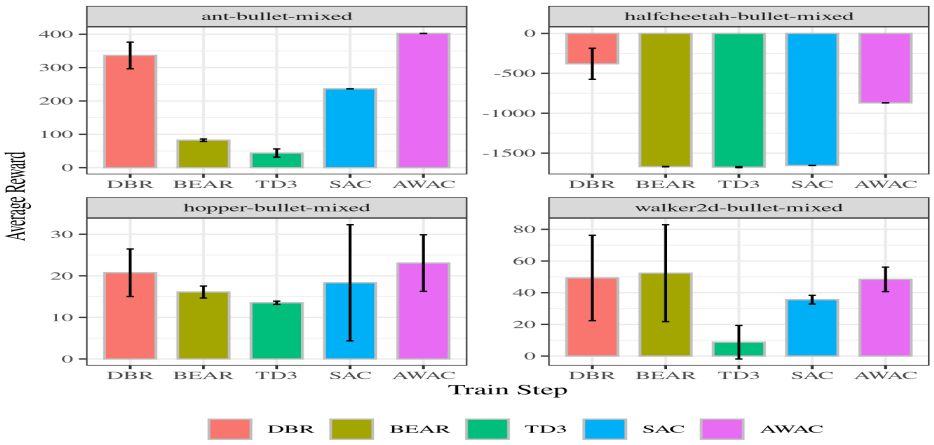
Figure 4.6 demonstrates the strength of our choice, and the degenerative behaviour present when using an implicit regularisation scheme as suggested by MRL. Furthermore as shown in Figure 4.2, that the strength of DBR is shown that superior policy can be obtained from a diverse set of experiences even if this includes lower-quality examples.



(a) “Medium” Offline Datasets

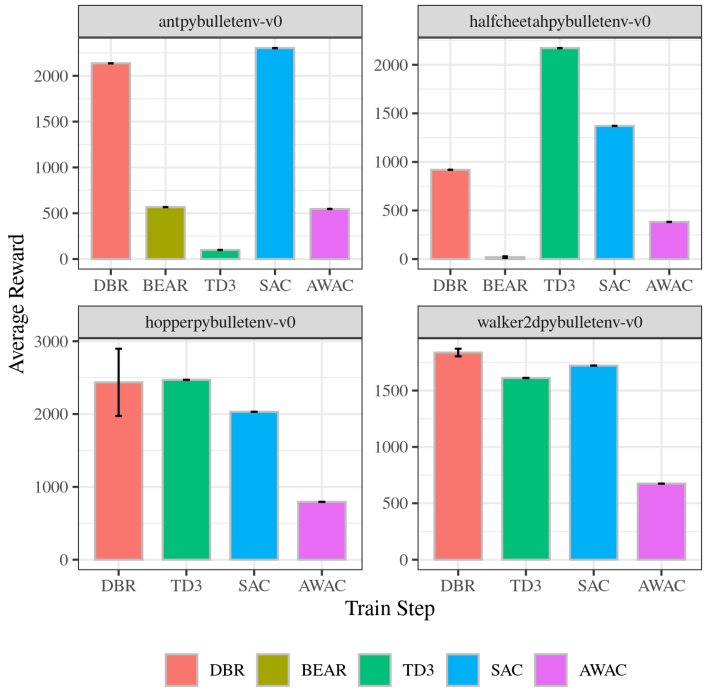


(b) “Random” Offline Datasets

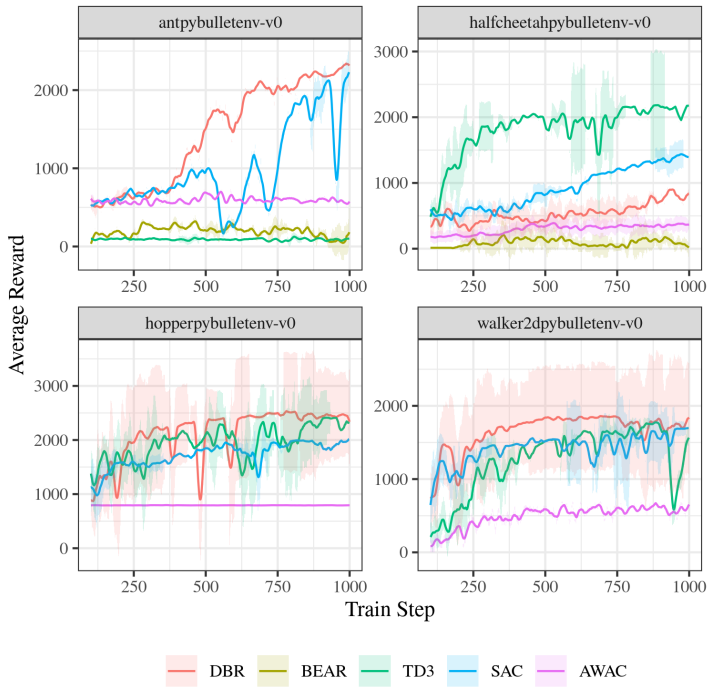


(c) “Mixed” Offline Datasets

Figure 4.4: PyBullet Control environments: each agent only used offline datasets with no exploration. (a) frozen policy after training (medium) (b) random policy, (c) mixture of random and medium quality experiences. The average is provided over the last ten steps, with error bars representing one standard deviation.

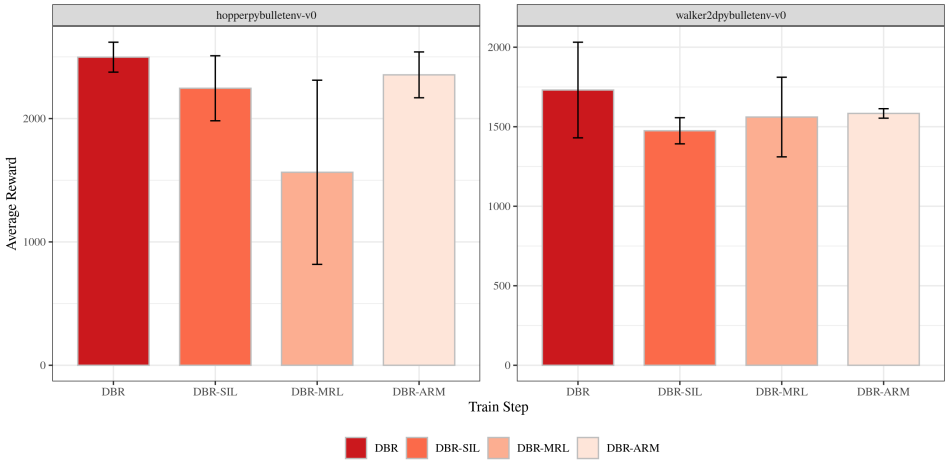


(a)

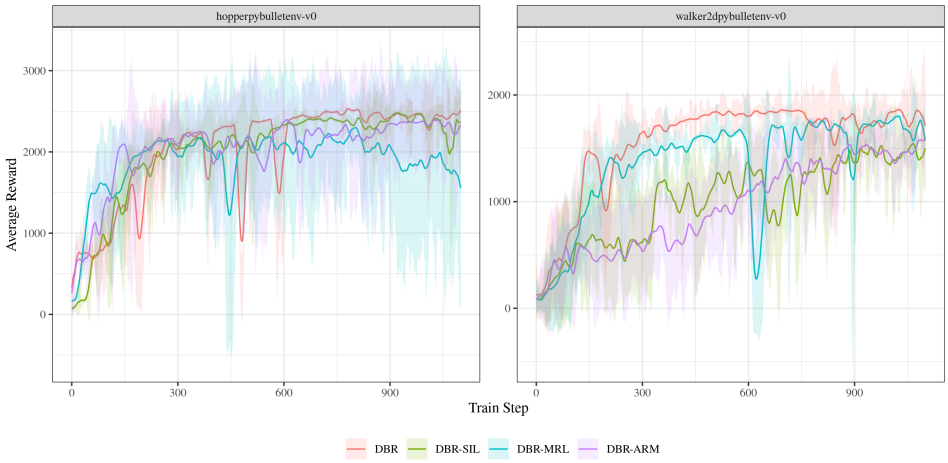


(b)

Figure 4.5: PyBullet environments, when trained tabula rasa (i.e. without offline data). The average is provided over the last ten steps, with error bars representing one standard deviation.



(a)



(b)

Figure 4.6: Ablation Results. The average is provided over the last ten steps, with error bars representing one standard deviation.

	DBR	Bear	IAC	SEAC
pursuit	19.76 ± 10.44	18.37 ± 9.77	16.86 ± 9.38	17.56 ± 9.69
reference	21.51 ± 9.94	7.54 ± 3.45	2.65 ± 1.28	2.68 ± 1.24
spread	7.60 ± 3.27	5.64 ± 2.46	3.78 ± 1.76	3.99 ± 1.87
tag	8.76 ± 3.68	5.65 ± 2.46	3.78 ± 1.76	3.99 ± 1.87
waterworld	21.25 ± 9.43	17.40 ± 8.36	15.17 ± 7.55	16.28 ± 7.75

Table 4.3: Final Evaluation returns with one standard deviation for multi-agent environments using DBR and BEAR as drop-in replacements for “shared experience actor critic” algorithm using PettingZoo environments.

4.5 Multi-Agent Experimental Results

The DBR framework can be extended to multi-agent scenario with great effect. In this section we perform ablation study and compare with other multi-agent actor-critic frameworks to determine the performance of DBR. As DBR makes efficient use of offline and off-policy information, we can drop-in DBR model in "shared experience actor critic" model which operates by enabling other agent experiences to be augmented with each agent in the CDTE MARL framework. To ensure consistency across tasks and environments, consistent preprocessing was used as suggested in other MARL benchmark [68]. When evaluating our agents, the tasks were run in separate evaluation environments with 1000 episode rollouts.

From the results shown in Table 4.3, we observe that DBR has superior performance compared with other actor critic approaches when used as a drop-in replacement for algorithms which can use additional history and data where the agents are assumed to be homogenous. This demonstrates the superior performance and opportunities provided by DBR algorithm.

4.6 Discussion and Future Work

In this chapter we formulated DBR, which is an approach to bridge offline reinforcement learning with off-policy approaches through relaxing constraints placed on it through the ARM framework and BEAR algorithm. This is achieved through removing the fixed constraint in the BEAR algorithm, and enabling DBR to dynamically control its exploration rate based on the advantage framework. This removal of the fixed constraint is central to our *detection* challenge in this part of this thesis, whereby it enables our algorithm to detect and adapt to the appropriate tasks based on the advantage framework.

This approach enables DBR to generalise over different types of tasks and datasets, including suboptimal policies as shown in the PyBullet experiments, and generalising over tasks such as Connect-4 where the training environment is simpler than the evaluation environment. We justified our choice for the construction of the clipped advantage both theoretically and empirically by comparing it with self-imitation learning and implicit regularisation in the Munchhausen RL framework.

Future extensions to our work could entail exploring more scenarios in which behaviours can be transferred from one context to another. Through exploring and researching these challenges, it can enable algorithms which can be trained using large scale datasets leading to similar progress made in recent years within the supervised learning fields in image recognition and natural language understanding.

5

Multi-Agent Regularised Q-Learning

Contents

5.1 Introduction

In the previous chapter we examined the challenge of detection in the single agent reinforcement learning (RL) scenario whereby we explored how we can learn a surrogate behaviour policy that is positively and negatively clipped to bridge the offline and off-policy algorithm which can generalise better and induce superior policies compared with the experiences that the policy may be based on.

In this chapter we continue with the *detection* challenge but instead in the multi-agent setting. Similar to the previous chapter we examine how we can improve our policies through optimising our available experiences and again also examine how explicit or implicit regularisation can be leveraged to improve multi-agent reinforcement learning (MARL). One key observation in MARL that algorithms leverage Q-learning is if we naively use experiences of other agents, it does not greatly improve performance [9]. In this chapter we explore *Multi-Agent Regularised Q-Learning* (MARQ), which strives to correct for the underlying policy distribution to enable experience sharing to occur through regularisation. Furthermore MARQ aims to address these challenges without explicit centralised structures which we hypothesis enable agents to perform structured and diverse exploration.

Unlike many other MARL approaches, MARQ does not leverage centralised structures [45, 47, 57, 76]. Instead, the MARQ framework is closer to *Independent Q-Learning* (IQL) [67] variation of *Shared Experience Actor-Critic* [9]. Despite lack of centralised structures, by imputing other agent's experiences, and correcting for policy distribution differences, MARQ can still attain superior performances compared with the analogous IQL approach. By having structured exploration, agents can leverage their collective experiences which allow agents to explore the most promising directions. This has been demonstrated empirically as MARQ generally yields greater returns with less iterations compared with other approaches.

In line with the challenge of *detection*, instead of using DQN MARQ uses a distributional reinforcement learning algorithm [12], which enables MARQ to apply importance sampling and divergence measure. This approach enables experiences to be gathered in an adaptive manner, and the direction in which agents perform exploration and the nature of the policy updates allow greater returns in less iterations.

In this chapter, we summarise our contributions to *Multi-Agent regularised Q-learning* as follows:

- We empirically demonstrate more efficient training by allowing of diverse experiences through regularisation compared with existing MARL Q-Learning approaches.

- We demonstrate that optimising over experience gathering and shared experience can outperform existing MARL Q-Learning approaches and independent learning approaches without expensive centralised structures.

5.2 Related Works

Diversification

To tackle the challenge of detection, one approach is to encourage diverse experiences to induce and find promising states. In single agent RL, this has been approached in an unsupervised manner by generating new skills through considering the maximum entropy without taking into account the reward signal [14]. The analogous scenario in MARL was constructed by considering mutual information in the joint state-action [47]. These approaches involve detecting and modelling the underlying policies involved. These are not the only approaches which can be used to promote diversification, as other approaches in RL exist which consider the exploitation and exploration trade-off [32, 56].

Entropy Regularisation

Entropy regularisation is an approach which can be leveraged to promote structured exploration. This is a common policy optimisation technique where an entropy penalty is added to the policy and value function update, and has been used in multiple single-agent reinforcement learning algorithms [26, 27, 60]. Entropy regularisation can also occur implicitly by augmenting the reward signal [70]. In the MARL scenario, an adaptive entropy regularisation approach has been empirically shown to promote consistent exploration across multiple agents [76].

5.3 Multi-Agent Regularised Q-Learning

Next we provide the necessary background on our approach "Multi-Agent Regularised Q-Learning" (MARQ). Then we outline our methodology and discuss the various considerations in the construction of the MARQ algorithm.

Multi-Agent Reinforcement Learning and Q-Learning

First we introduce our notation and provide background to multi-agent reinforcement learning as part of the MARQ setup. Within MARQ, the problem can be framed as a Dencentralised Partially Observable Markov Decision Process [53]. This is defined by the tuple $\langle S, U, P, R, Z, O, n, \gamma \rangle$ where the state of the environment is $s_t \in S$ where t represents the time step embedded in the environment. Then for every time step t, each individual agent $a \in A \equiv \{1, \ldots, n\}$ selects their corresponding action $u \in U$ independently from all other agents. We define the joint action of all agents to be $\mathbf{u} \in \mathbf{U} \equiv U^n$. As part of the environment, it contains a probability transition function or the state transition function $P(s_{t+1}|s_t, \mathbf{u}_t)$. In our experiments, we restrict the MARL problem to be cooperative, that is, the same reward function and reward is provisioned to all agents and

defined by $R(s_t, \mathbf{u}_t)$ with discount rate $\gamma \in [0, 1)$.

In this thesis, environments are presumed to be partially observable, we formalise this by restricting each agent to their own local, individual observation $z \in Z$ based on the observation function $O(s, a) \rightarrow Z$. Then by convention, a given particular agent a's action-observation history is defined as τ^a, so that this conditions on the agent's stochastic policy $\pi^a(u^a|tau^a)$.

In this formulation, the Q function for a particular agent i is defined as

$$Q^i(s_t, u_t^i) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}^i}\left[R_t|s_t, u_t^i\right] \tag{5.1}$$

Then in Independent Q-Learning [67], the objective is the Bellman optimality equation which is as follows

$$Q^*(s, u) = \mathbb{E}_{s'}\left(r + \gamma \max_{u'} Q^*(s', u'|s, u)\right) \tag{5.2}$$

For a particular agent a, The Q function would be approximated through a neural network function approximator $Q(s, u; \theta^a)$, where the algorithms typically leverage parameter sharing across agents for efficiency. The object of this approach is to minimise the following loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{s,u,r,s'}\left((y - Q(s, u; \theta^a))^2\right) \tag{5.3}$$

Where $y = r + \gamma \max_{u'} Q(s', u'; \theta^-)$ with θ^- being the parameter of a target network. The target network is frozen for several training steps and is then updated according to a predetermined schedule with the baseline Q function. A distributional variation for Q learning is created through having the Q function outputting the distribution estimates and using quantile Huber Loss instead of the mean squared error returns when modelling the Q function [12].

Another extension over the distributional variation of Q learning is *Conservative Q-Learning* (CQL). If the empirical behavioural policy at state s is defined as $\hat{\pi}_{\mathcal{D}}(u|s) := \frac{\sum_{s,u \in \mathcal{D}} \mathbf{1}[s=s,u=u]}{\sum_{s \in \mathcal{D}} \mathbf{1}[s=s]}$, the addition of Q-value *maximisation* term (shown in red) from $\hat{\pi}_{\mathcal{D}}$; below is the iterative update at timestamp t:

$$\hat{Q}_{t+1} \leftarrow \underset{Q}{\operatorname{argmin}}\, \alpha \cdot \left(\mathbb{E}_{s \sim \mathcal{D}, u \sim \pi(u|s)}[Q(s,u)] - \mathbb{E}_{s \sim \mathcal{D}, u \sim \hat{\pi}(u|s)}[Q(s,u)] \right) + \frac{1}{2} \mathbb{E}_{s,u,rs' \sim \mathcal{D}} \left((y_t - Q(s,u;\theta)^2 \right) \tag{5.4}$$

CQL provides theoretical improvement guarantees over the lower bound, and provide extensions through the choices of regularisers $\mathcal{R}(\pi)$. In this chapter, we are interested in the multi-agent setting, where there are additional considerations when leveraging other agent experiences.

5.3.1 Correcting for Shared Experiences

In constructing MARQ, we use the *Independent Q-Learning* framework distribution Q-learning approach instead of DQN. Prior work has shown that using shared experience in Q-Learning frameworks has demonstrated to have minimal improvement in performance in the MARL setting[9]. In this section, we describe how to improve and rectify the deficiencies which were identified.

The key observation is if we leverage Q-learning and shared experiences without correction, this may cause the Q function to be underestimated.

Proposition 5.1. *If the DQN iterative update for agent i using the observation history from agent k is provided as follows:*

$$\hat{Q}^i_{t+1} \leftarrow \underset{Q}{\operatorname{argmin}}\, \alpha \cdot \mathbb{E}_{s \sim \mathcal{D}^k, u \sim \pi^i}[Q(s,u;\theta^i)] + \frac{1}{2} \mathbb{E}_{s,u \sim \mathcal{D}^k} \left[(y^i - Q(s,u;\theta^i))^2 \right] \tag{5.5}$$

Then the resulting Q-function $\hat{Q}^k := \lim_{t \to \infty} \hat{Q}^i$, lower bounds $\hat{Q}_k$ at all (s,u).

Proof. Consider the iterative update Equation 5.5 above. If the derivative is set to 0, it yields the below equation:

$$\forall s,u \in \mathcal{D}^k,\ \forall t \quad \hat{Q}^i_{t+1}(s,u) = y^i_t - \alpha \frac{\pi^i(u|s)}{\pi^k(u|s)} \tag{5.6}$$

Observe that $\pi^i(u|s) > 0, \pi^k(u|s) > 0, \alpha > 0$. When we consider the iterative update, the Q-value will be underestimated, i.e. $\hat{Q}^i_{t+1}(s,u) \leq y^i_t$. The intuition here is without adjustment, the experience realised from another agent may be inappropriate. If we consider all pairwise combinations of agents, we can extend this to all agents.

To improve this setup, we can attempt to correct through the value loss function. If the shared experience is an out of distribution sample, the value loss would be high due to undersampling. This leads to the addition of a *regularisation penalty* to the value loss component similar to the *Shared Experience Actor-Critic* approach [9]. We demonstrate this below, where using our notation define the update iterate for agent i by sampling from agent k to be $\hat{Q}^{\{i,k\}}_{t+1}$, and λ being a hyperparameter which weighs the strength or reliance on experiences from other agents; with the new regulariser component shown in red:

$$\begin{aligned}\hat{Q}^{\{i,k\}}_{t+1} \leftarrow \underset{Q}{\operatorname{argmin}}\, & \alpha \cdot \left(\mathbb{E}_{s\sim\mathcal{D}^k, u\sim\pi^i}[Q(s,u;\theta^i)] - \mathbb{E}_{s\sim\mathcal{D}^k, u\sim\hat{\pi}^k(u|s)}[Q(s,u;\theta^i)]\right) \\ & + \frac{1}{2}\mathbb{E}_{s,u\sim\mathcal{D}^k}\left[(y^i_t - Q(s,u;\theta^i))^2\right] + \lambda \cdot \mathbb{E}_{s,\cdot,r,s'\sim\mathcal{D}^k, u\sim\pi^i}\left[\frac{\pi^i(u|s)}{\pi^k(u|s)}\right]\left\|V^i(s) - \left(r + \gamma V^i(s')\right)\right\|\end{aligned} \tag{5.7}$$

Similar to *Shared Experience* framework [9], we have found the hyperparameter λ to be largely insensitive, and have used $\lambda = 1$ in our experiments with a separate ablation study to justify our reseults in Section 5.4.2.1. Our experiments and setup are different from Christianos et al (2020) [9], as we leverage a distribution RL approach which allows for importance sampling and for the correction to be used. This demonstrates how a

Q-learning approach can be used to align performance with the results in the actor-critic framework.

5.3.2 Explicit Adaptive Regularisation

Our approach is not the first approach to explore the impact of regularisation for MARL algorithms. As part of *Learning Implicit Credit Assignment* (LICA), structured exploration was encouraged through the addition of an entropy regulariser. This is adaptive entropy was created dynamically by controlling the entropy gradients to ensure they are consistent across all agents during the training step

$$\mathcal{H}^i := \kappa H(\pi^i)$$
$$\partial\mathcal{H}^i := -\kappa \cdot \frac{\log \pi^i + 1}{H(\pi^i)}$$

where the entropy is H, policy of agent i is π^i, and the hyperparameter to control regularisation strength is κ. In the MARQ framework, we are operating using Q-Learning approach, and not the actor-critic approach which is used in LICA. Instead a distributional RL approach is taken which can allow for the inclusion of the entropy regularisation. Rather than having a regulariser which operates only on each agent independently as in LICA, diverse trajectories are encouraged through the cross-entropy regulariser. Furthermore the advantage in using this approach is that it does nor require the reward signal and can be computed in an unsupervised manner. With cross entropy defined as $H(p,q) = H(p) + D_{\mathrm{KL}}(p\|q)$, the iterative Q-learning update from Equation 5.8 the regulariser penalty (shown in red) is:

$$\hat{Q}_{t+1}^{\{i,j\}} \leftarrow \underset{Q}{\operatorname{argmin}}\, \alpha \cdot \Big(\mathbb{E}_{s\sim\mathcal{D}^i,u\sim\pi^i(u|s)}[Q(s,u;\theta^i)] - \mathbb{E}_{s\sim\mathcal{D}^i,u\sim\hat{\pi}^i(u|s)}[Q(s,u;\theta^i)]\Big)$$
$$+\frac{1}{2}\mathbb{E}_{s,u,rs'\sim\mathcal{D}^i}\left[(Q(s,a)-y_t^i)^2\right] + \kappa \sum_{k\in\{1,\ldots,n\}} \Big(H(\pi^i) + D_{\mathrm{KL}}(\pi^i \| \pi^k)\Big) \tag{5.8}$$

Where κ controls the regulariser. One observation in this setup is since cross entropy by definition is non-negative, it will encourage diversity which is offset by the entropy correction. This demonstrates how we can extend LICA's adaptive entropy.

5.4 Experiments

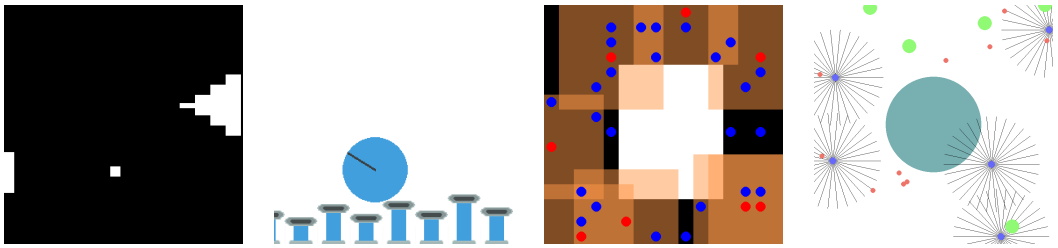


Figure 5.1: Petting Zoo Environments: pong, pistonball, pursuit and water world.

Our MARL experiments are conducted through the benchmark MARL datasets [25, 45, 68] depicted in Figure 5.1.

Cooperative Pong is the original Atari Pong but provided as a multi-agent variation [68]. In this cooperative game, the goal is to ensure the ball stays in play. This is achieved through allowing the agents to "guard" the left and right side of the field by only being able to move vertically.. To make this task challenging, the agents can be set to be heterogenous, in which one of the paddles is shaped as a tiered cake. The reward structure is a dense, whereby a positive reward is provided if the game is in progress otherwise a terminal, negative reward is provided.

Pistonball is a cooperative game where there are multiple pistons each controlled by its own agent [68]. The objective of this task is to move a ball from one end of the

environment to the other through activating pistons which can move up and down. In this scenario, coordinated behaviour is necessary across agents to ensure success. The reward structure is based on the movement of the ball.

Pursuit is an environment where we take control of purser agents, with the aim to capture randomly moving evaders [25]. In this task, rewards are provided whenever an evader is captured and subsequently removed from the environment.

Waterworld is an environment that comprises of archea attempting to navigate and survive in a simulation [25]. The task in this scenario is to find food and avoid poison, that can generate positive and negative rewards accordingly.

In the *Spread* and *Reference*environments, the environment consists of stationary landmarks in which we can leverage multiple agents to interact with the environment [45]. In both environments, the goal is for the agents to move closer to landmarks. Both tasks have the same action space, but the objects differ slightly. In *Reference*, each agent has a target landmark they are to approach, but not directly observable to themselves, but is known to the other agents, and have additional actions which enable direct communication with other agents. In *Spread* all agents must approach all other landmarks without collision. In these environments, the reward is provided based on distance to the landmarks, with Spread providing penalties if agents collide.

5.4.1 Experiment Details

We use consistent parameters and preprocessing across all environments, using defaults suggested by MARL benchmarks [68]. During evaluation, we run on a separate environment with 1000 evaluation steps.

In our experiments, we used the setup and code based on official implementations. The pymarl library was used for LICA, QMIX, IQL, whilst rlkit was used to reimplement SEAC, MADDPG with gumbel softmax [34, 46] support for discrete action space for

Policy Hidden Shape	[64, 64, 64]
Mixer Hidden Shape	[32, 32, 32]
Activation Function	ReLU
Learning Rate	0.0003
Replay Buffer Size	1000000
Steps per Iteration	1000
Discount	0.99
Reward Scale	1

Table 5.1: The MARL Experiment hyperparameters

SEAC and MADDPG, which is inline with the approach suggested by Christianos et al (2020) [9]. In all models, parameter sharing was used as in the pymarl setup.

5.4.2 Results

In this section we compare MARQ against existing state of the art approaches. Through our results, we demonstrate MARQ has comparable if not stronger performance compared with existing approaches (Figure 7.5). In addition to our strong performance, MARQ also requires less training time compared with other approaches which used a centralised mixing network. Our approach does have an increased training time compared with IQL due to the distributional RL base algorithm which is used.

5.4.2.1 Ablation

For our ablation studies, we examine different hyperparameters and choices for our regulariser. Most importantly our approach is not sensitive to the shared experiencing weighting irrespective of the regularisation approach used. More importantly we demonstrate that as $\lambda \rightarrow 0$, the model approaches a model which does not leverage shared experiences and leads to a decreased performance for the approaches using shared experience KL divergence, shown in Figure 5.4 and 5.5.

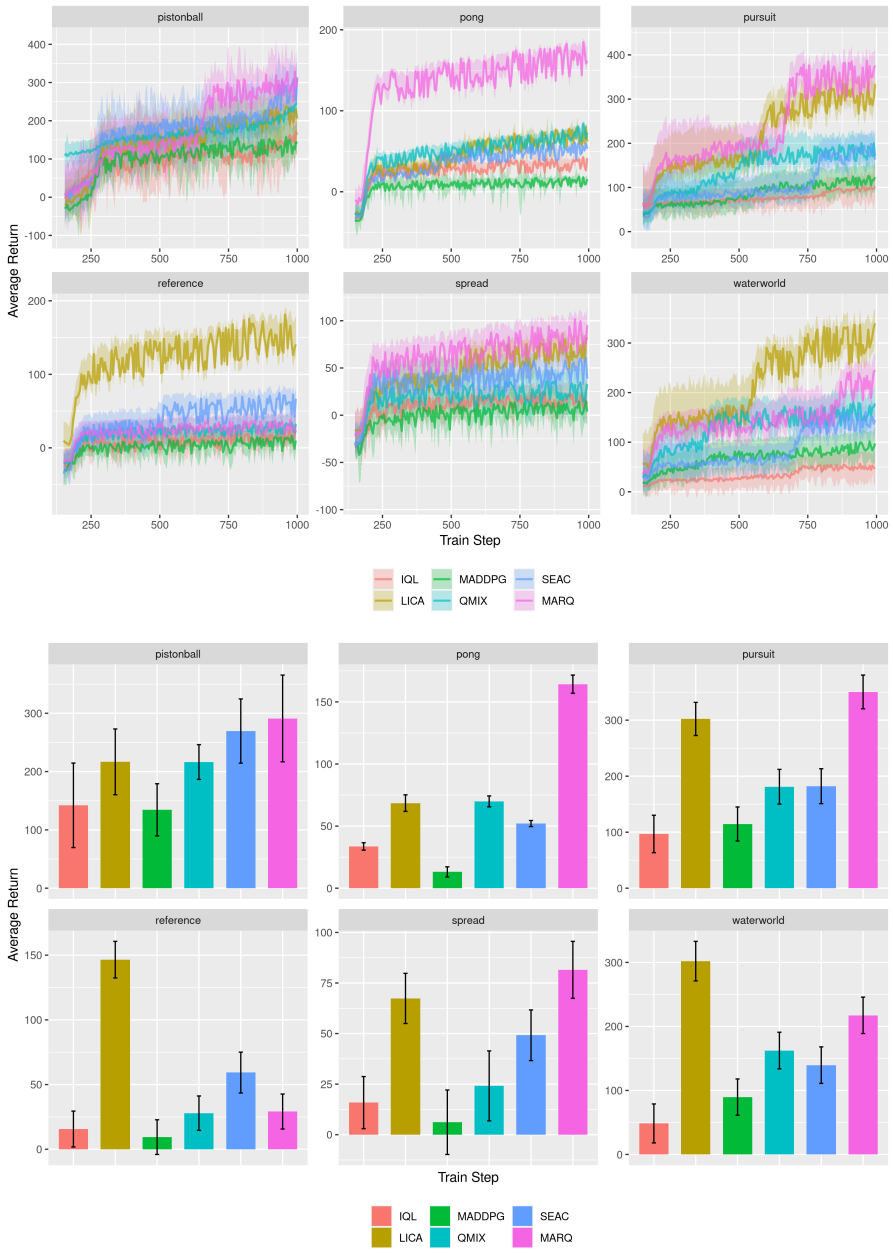


Figure 5.2: Comparison over a variety of benchmark MARL tasks. The average is provided over the last ten steps, with error bars representing one standard deviation.

5.5 Conclusion

In this chapter, we have explored the Multi-Agent Regularised Q-Learning approach, which demonstrates effective ways to leverage shared experiences in Q-Learning with adjustments to ensure policy distributions are appropriately catered for. This tackles the

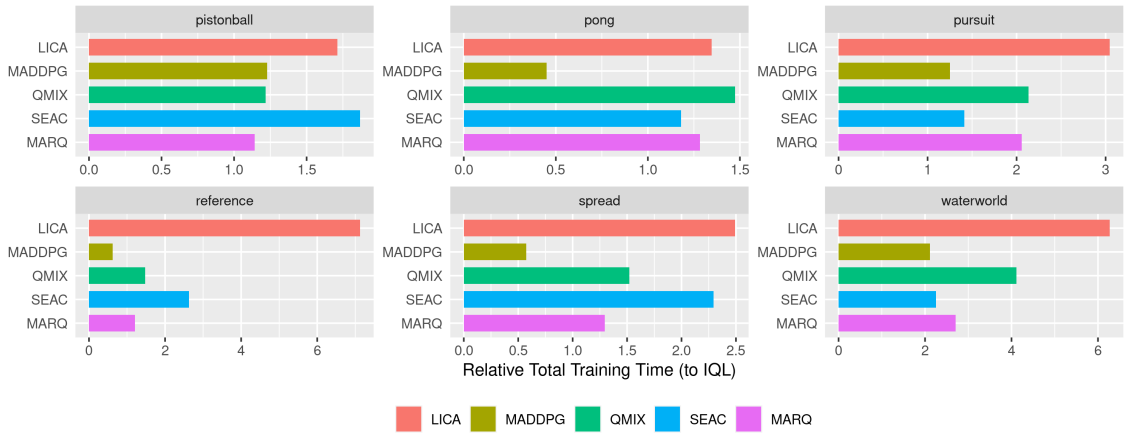


Figure 5.3: Training time evaluation using IQL as the baseline.

detection challenge laid out in this thesis by being able todetect and adapt to changing circumstances provided to us in the shared experience paradigm.

This approach has excellent speed of convergence and strong performance compared with other approaches despite lack of the centralised mixing structures of centralised algorithms. Through our empirical results, we have justified our algorithm, including the neural architecture decisions which reveal the stability of our approach.

Further avenues of research could explore improved ways to tune the amount of sharing depending on the MARL environment, examining whether cross entropy regularised penalties help or hinder actor-critic frameworks and greater consideration of the role of diverse states and exploration strategies in MARL.

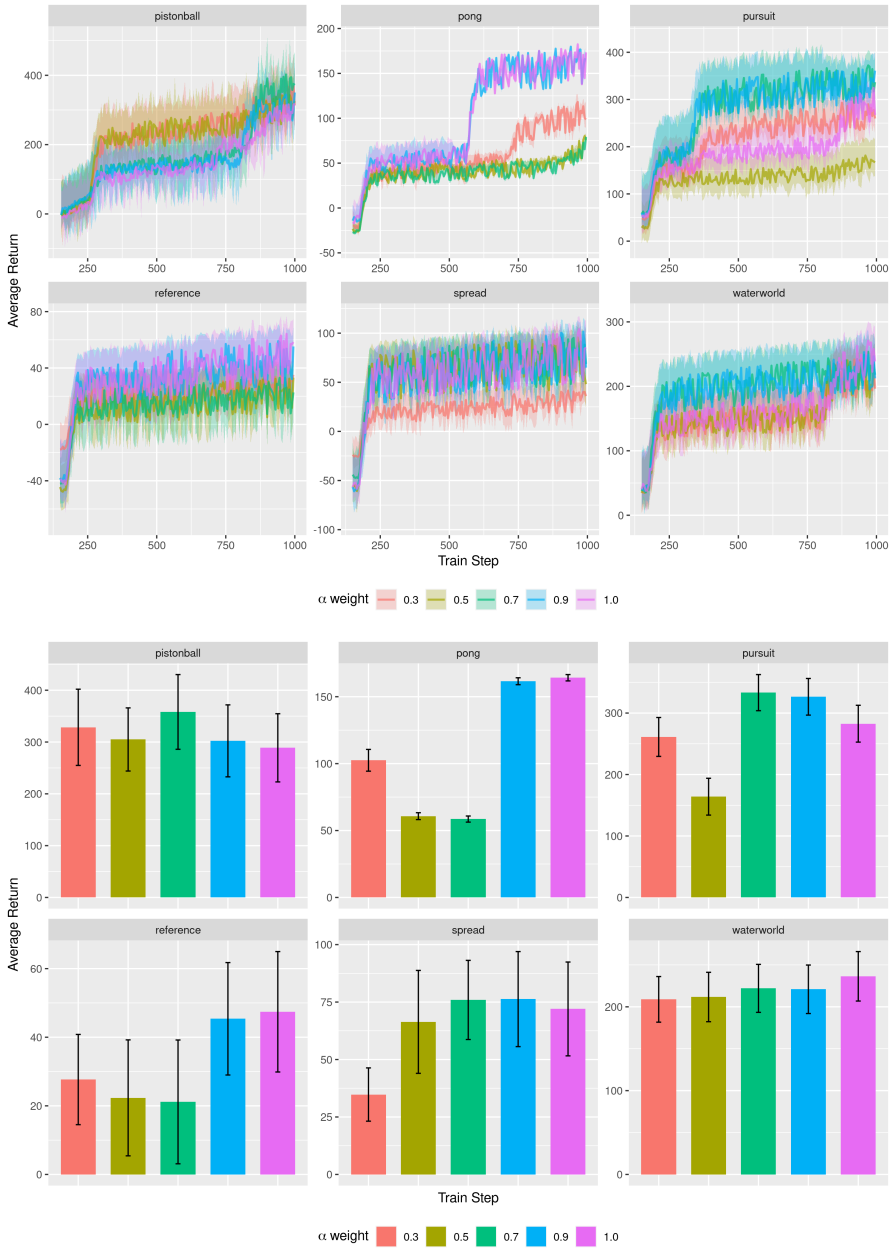


Figure 5.4: Sensitivity analysis hyperparameter λ with explicit mixing. The average is provided over the last ten steps, with error bars representing one standard deviation.

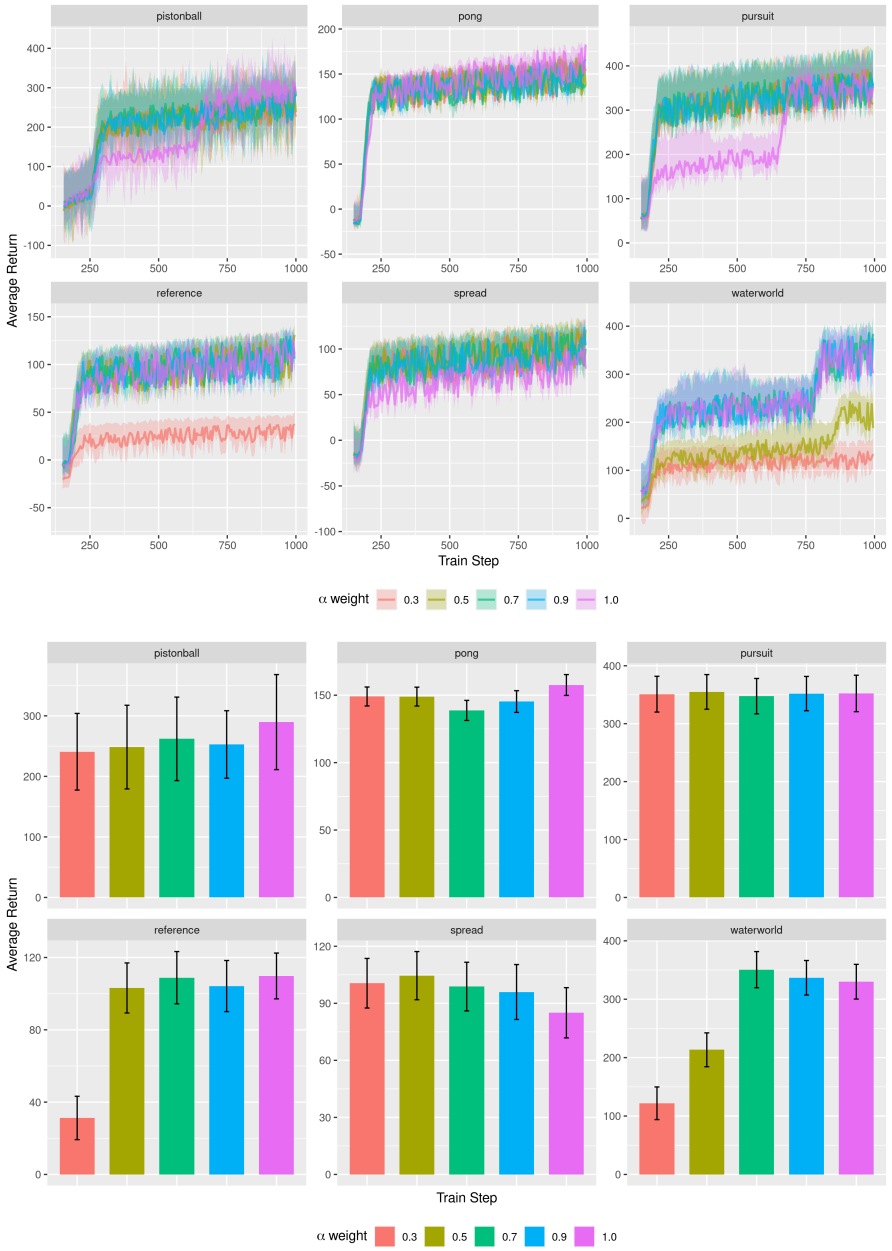


Figure 5.5: Sensitivity analysis hyperparameter λ with implicit mixing. The average is provided over the last ten steps, with error bars representing one standard deviation.

Part III

Learning to Collaborate

Abstract

In this part of the thesis we focus on the topic of *collaboration*. To achieve this we first consider how we can explore neural architectures in chapter 6, and then its application to multi-agent reinforcement learning in chatper 7. From the multi-agent perspective, we examine how we can leverage multiple agents to generate diverse experiences by allowing agents to decouple from centralised structures with minimal overhead allowing our approaches to adapt to different MARL environments which may benefit from different levels of cooperation, and demonstrate our approach empirically.

6

Residual Networks Act Like Boosting Algorithms

Contents

6.1 Introduction

In this chapter we tackle the problem of *collaboration* through the lens of exploring and choosing neural architecture choices. In particular we consider Residual Networks (ResNet) [31], and demonstrate the equivalence to boosting feature representations, and how they can be extended to AdaNet framework and retain the theoretical bounds offered to it via Online Gradient Boosting Theory. We will use this approach to justify our neural architecture choices for MARL neural network architecture selection by interpreting an ensemble classifier as a special variant of mixer network as introduced by QMIX.

Prior to our work, there have been other attempts to explain ResNets in various ways including:

- Attempting to unravel ResNet itself [69]
- Observing the connection between identity loops and its ability to avoid local optimas [28]
- Treating residual modules as weak classifiers which can be induced through sequential training [33]

One reason why there is such a strong interest in interpreting Resnets, is due to the empirical evidence and the subsequent architectures which have empirically been easier to optimise, and consistently have demonstrated superior performance on a variety of datasets including ImageNet and CIFAR-10 [31].

In this chapter, we first examine ResNet and Online Boosting, highlighting the conditions in which they are equivalent. This is shown through observing that layer by layer boosting in ResNet can be represented through additive modelling approaches

in Logistboost [16], with the key difference being that the feature representations are boosted rather than the label itself. Specifically, if we leverage a composite loss function in the Online Boosting framework, then it is the same as ResNet. As a consequence, ResNet does not adhere to traditional boosting theory results. This is because it cannot be represented as a linear weighted ensemble, however through the results from the Online Boosting framework through the regret bounds apply to ResNet architectures.

Armed with this intuition, we explore further different ways in which the ResNet architecture can be updated using ideas from Online Boosting. This includes adding a dynamic, *learnable* shrink parameter, rather than a fixed hyperparameter which is used in ResNet, offering additional flexibility to the ResNet framework.

Here, the vanilla ResNet approach is compared with our modified approach on benchmark dataset CIFAR-10 and Street View House Number task, whereby our approach demonstrates some improvement over the vanilla approach with the addition of the shrinkage parameters.

Next we reinterpret ResNet to the supervised tabular setting by implementing boosted decision trees as a neural network, and evaluate our results on multiple supervised learning benchmark datasets and other tree ensemble methods being *Deep Neural Decision Forests* [39], *Adanet* [11] (using decision trees with decision tree modules) and Gradient Boosted Tree algorithms. In our experiments, our approach produced greater performance to the other neural network variants compared with the tradition gradient boosted tree ensemble models. Finally we apply this architecture to the RL and MARL environment benchmark datasets to demonstrate its ability to generalise beyond supervised learning problems.

6.2 Preliminaries

We provide preliminary background for the rest of the chapter, covering ResNet and boosting. We will examine the regret bounds formed as part of the Online Gradient Boosting setting [4] and AdaNet generalisation bounds [11].

6.2.1 Residual Neural Networks

In this section we provide an overview of the ResNet algorithm. ResNet is comprised of stacked "residual blocks". These modules are composed of an identity function $f_u(x)$, where each identity function call maps its input, where the depth of the module is denoted by u. Here, $f_u(x)$ is any kind of sequential neural network layers, such as convolutions (in image recognition). In this chapter, we examine image recognition setup using convolutions and also how decision tree modules can be constructed to build a boosted neural network decision tree representation.

Let $g_{u+1}(x)$ be the output of the u-th residual block with input x in a ResNet model with corresponding output:

$$g_{u+1}(x) = g_u(x) + f_u(g_u(x)) \tag{6.1}$$

Output is then defined recusively as in Equation 6.1 with the U-th residual moduled constructed by summing prior modules together. That is $g_{U+1}(x) = \sum_{u=0}^{U} f_u(g_u(x))$, with the initial conditions $g_0(x) = 0$, $f_0(g_0(x)) = x$. Afterwards, a linear layer is used with the output of the appropriate size for the supervised learning problem at hand.

$$\hat{y} = \widehat{\sigma}(\mathbf{w}^\top g_{U+1}(x)) \tag{6.2}$$

$$= \widehat{\sigma}\left(\sum_{u=0}^{U} \mathbf{w}^\top f_u(g_u(x))\right) \tag{6.3}$$

where $\widehat{\sigma}$ is a mapping of classification outputs to the appropriate label, which is typically chosen to be a sigmoid function like softmax.

6.2.2 Boosting

In this section we provide an overview of boosting. Boosting is an approach to combine weaker learners, which is any model that performs better than a naive model, to a strong learner. In AdaBoost, the algorithm chooses training sets to construct weak classifiers to create novel inferences [16]. This style of boosting follows the approach in BoostResNet [33]. Whereas for gradient boosting, pseudo-residuals are used for training and can also operate in the online setting [4].

If we define $h_u(x)$ to be input function of x with α_u being the weight for u-th model, then we can reframe boosting as $\widehat{F}(x) = \sum_{u=1}^{U} \alpha_u h_u(x)$ which is a linear combination of U models [16] [4]. To boost using the feature representation input, logitboost was introduced [16]; in this scenario boosting leverages the softmax (or logit) transform

$$\hat{y} = \widehat{\sigma}(\widehat{F}(x)) \tag{6.4}$$

$$= \widehat{\sigma}\left(\sum_{u=1}^{U} \alpha_u h_u(x)\right) \tag{6.5}$$

Where $\widehat{\sigma}$ is the logit or softmax function. As we can see, the ResNet linear classifier layer algorithm possesses a similar form. Furthermore, this approach can be extended to the online formulation. To achieve this, one adjustment is required. First, the partial sums defined by $\hat{y}^{i-1}$ is updated to include a shrinkage parameter which is used to dynamically update the weights of each module, which is updated via gradient descent. To ensure convergence, the outputs are must be bounded [4].

One important distinction between online gradient boosting and traditional boosting learning bounds is that it is regret based. The difference between best hypothesis learner and the loss from the learner is defined as:

$$R_{\mathcal{A}}(U) = \sum_{u=1}^{U} \ell_u(h_u(x_u)) - \min_{h^* \in \mathcal{F}} \sum_{u=1}^{U} \ell(f^*(x_u))$$

This regret bound used in online gradient boosting, applies to weak linears that could be combined linearly. This is subject to weak learners which have a linear, convex, loss function which is 1-Lipschitz bounded.

Corollary 6.1. *(Based on Beygelzimer et Al. 2015) With* $\eta \in [\frac{1}{N}, 1]$ *being the learning rate, N representing the weak learners, both which are hyperparameters. Then Algorithm 6.1 for* $span(\mathcal{F})$ *is an online learning algorithm, provided the loss functions are convex, linear and Lipschitz constant which is bound by* 1*. This would have the regret bound, so that for any* $f \in span(\mathcal{F})$*:*

$$R'_f(U) \leq (1 - \frac{\eta}{||f||_1})^N \Delta_0 + O\left(||f||_1 \cdot (\eta U + R(U) + \sqrt{U})\right)$$

where $\Delta_0 := \sum_{u=1}^{U} \ell_u^{\psi}(0) - \ell_u^{\psi}(f(x_u))$ *this is the initial error, with the regret for the base learner defined as R.*

For this theorem to hold, there are two conditions which need to be satisfied. First, the weak learner $\mathcal{A}$, must be bounded, that is $\|\mathcal{A}(x)\| \leq D$, for a fixed, pre-determined value D. Second, the loss functions chosen must have a computable subgradient which possess a finite upper bound. Consequently, the constraints placed by BoostResNet which requires the explicit distribution to be maintained over the training set is removed when leveraging the online boosting formulation. Next we consider AdaNet Generlisation Bounds as an additional interpretation of ResNets.

In feedforward neural networks, the AdaNet Generlisation Bounds is a multi-layer architecture, which acts as a directed acyclic graph [11]. In this formulation, there

are two conditions which must be satisfied. First, all layer weights are bounded with l_p-norm, where $p \geq 1$. Second, the activation function in the feed forward network is bounded Lipschitz constant by 1. Under these constraints, generalisation error bounds shown below holds [11]:

Corollary 6.2. *(Based on Cortez et al 2019) Define the distribution over $\mathcal{X} \times \mathcal{Y}$ to be $\mathcal{D}$ and a sample of m examples to be defined by S which are chosen indepdently at random based on $\mathcal{D}$. Then with $F(x)$, our decision tree classifier, would satisfy with probability at least $1-\delta$:*

$$R(f) \leq \hat{R}_{S,\rho}(f) + \frac{4}{\rho}\sum_{k=1}^{l}|w_k|_1\mathcal{R}_m(\widehat{\mathcal{H}}_k) + \frac{2\sqrt{\frac{\log l}{m}}}{\rho} + C(\rho, l, m, \delta)$$

$$\text{with } C(\rho, l, m, \delta) = \sqrt{\lceil \frac{4}{\rho^2}\log(\frac{\rho^2 m}{\log l})\rceil \frac{\log l}{m} + \frac{\log\frac{2}{\delta}}{2m}}$$

Notice that this bound is dependent on the (logarithmic) depth in relation to the network l. Based on the network l, earlier layers would have a much greater effect than the latter layers of the neural network. In the next section, we explore ways to construct novel neural network architectures by examining the formulation of ResNet and ideas from various boosting approaches.

6.2.3 Related Works

Due to the empirical success ResNet has had, many researchers have tried to explain how ResNet yields its results. One approach is to reinterpret ResNet as a "multi-channel telescoping sum boosting" framework which enables sequential training that creates the BoostResNet algorithm [33]. Another attempt focused around ResNet under linear constraints which sought to understand its representational power through observing importance of the identity connections and the ability to avoid local minimas [28]. In Highway networks, they considered the identity layers are shortcuts through the network

to carry gradients through that provides intuition for how the paths enabled the network to behave like a shallow ensemble [69].

Of particular interest to our approach is the relationship between BoostResNet [33] and AdaNet [11]. In both of these papers, they provide architectural changes to the underlying neural network to justify the *usage* of boosting, instead we aim to explore ResNet without altering how ResNet is trained or the ResNet architect itself.

To understand the changes to the ResNet, first, the sequential nature of the BoostResNet algorithm means that the model can never be truly updated online. For AdaNet, additional feature vectors in arbitrary formats are sequentially added which may not be compatible with the ResNet structure. Furthermore, vanilla AdaNet promotes a "bushy" representation, rather than a deep feature representation that exists in ResNet. We demonstrate that ResNet can be interpreted as a special case of the AdaNet framework, where we stack modules deep rather than wide, which means that ResNet possess AdaNet generalisation guarantees as they belongs to the same category of feedforward neural networks. To emphasis this, we create Neural Decision Tree ResNet modules to be used in the supervised scenario compared with Gradient Boosted Tree algorithms and compare their performances.

6.3 On the Equivalence of Boosting and ResNet

Recall from equations 6.2 and 6.4, LogitBoost and ResNet clearly have their similarities. Ultimately, both formulations both leverage the feature representation aspect of the neural network in its boosting. In ResNet, a linear classifier $\mathbf{w}$ connected across all modules.

Assumption 6.1. *Let $\widehat{h}_u(x) := \mathbf{w}^\top f_u(g_u(x))$ to be the u-th residual module, where $\mathbf{w}$ is a trainable. The residual module $\widehat{h}_u(x)$ is said to be a hypothesis module or weak learner*

$\forall u \geq 0$.

The weak learner condition is required to ensure as per Corollary 6.1 the learning bounds are upheld. Next we demonstrate the different variants of ResNet modules which are the basis for the experiments. All these variations follow the weak learning condition that is required for Online boosting guarantees to hold.

6.3.1 Online Boosting Considerations

Next we consider regret bound analysis specifically for in ResNet formulation and demonstrate the regret bounds still hold for a base weak learner that uses linear loss functions which are convex with 1-Lipschitz condition.

Algorithm 6.1 Composite Loss Functions for span($\mathcal{F}$) variation - Online Boosting

Copy a given algorithm $\mathcal{A}$ N times, which we allow it to be: $\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^N$, with hyperparameter $\eta \in [\frac{1}{N}, 1]$ being the step size
for $i = 1, 2, \ldots, N$ **do**
  set $\theta^i = 0$
end for
for $u = 1, 2, \ldots, U$ **do**
  Sample $\mathbf{x}_u$
  Define $\mathbf{F}_u^0 = 0$
  for $i = 1, 2 \ldots, N$ **do**
    $\mathbf{F}_u^i = (1 - \theta^i)\mathbf{F}_u^{i-1} + \eta \mathcal{A}^i(\mathbf{x}_u)$, with shrinkage $(1 - \theta^i)$ for $\mathcal{A}^i$
  end for
  Determine the output of the learner $\mathbf{y}_u = \psi^{-1}(\mathbf{F}_u^N)$
  Calculate $\ell_u^{\psi}(\mathbf{F}_u)$
  for $i = 1$ to N **do**
    Using the calculated loss, pass to $\mathcal{A}^i$ via $\mathbf{F}_t^{i-1}$
    Update $\theta^i \in [0, \eta]$
  end for
  Evaluate algorithm using $\mathbf{y}_u$
end for

Through Algorithm 6.1 and Corollary 6.1, our algorithm follow its regret bounds. Next, we need to check the composite loss functions satisfies conditions as described in Section 6.3.2.

6.3.2 Analysis

In Corollary 6.1, we consider the set of convex linear loss defined by $\mathcal{C}$. We examine considitions so that a composite loss function $ell^{\psi} \in \mathcal{C}$.Reid et al. (2010) [58] has shown that a sufficient condition for a composite loss function to belong in $\mathcal{C}$ is if it is a composite function with its canonical link.

Lemma 6.1. *Smoothness is retained in composite loss functions*

Proof:

Proof. Support ψ^{-1} is a Lipschitz continuous function and bounded. Then it follows that the composite function is also Lipschitz bounded. This can be shown below

$$\begin{aligned} |f_2(f_1(x)) - f_2(f_1(y))| &\leq L_2 |f(x) - f_1(y)| \\ &\leq L_1 L_2 |x - y| \end{aligned}$$

■

Consequently if ψ^{-1} has Lipschitz constant bounded by 1 and the composition is also bounded by 1, then the resulting composite loss function belongs to $\mathcal{C}$. Consider the cross entropy loss. In this scenario, the canonical link for cross entropy is softmax. As this has Lipschitz constant of 1, then the composite loss is also in class $\mathcal{C}$. Therefore ResNet satisfies the regret bound has it has a logic link and boosts the feature representation of a neural network.

One approach to convert BoostResNet to online learning is to leverage a common linear layer across all ResNet submodules. Then as in Algorithm 6.1 line 12 would be resolved as it would enable the online boosting algorithm update step to compute reference boosting function's gradient. Notice in BoostResNet side-steps this and instead

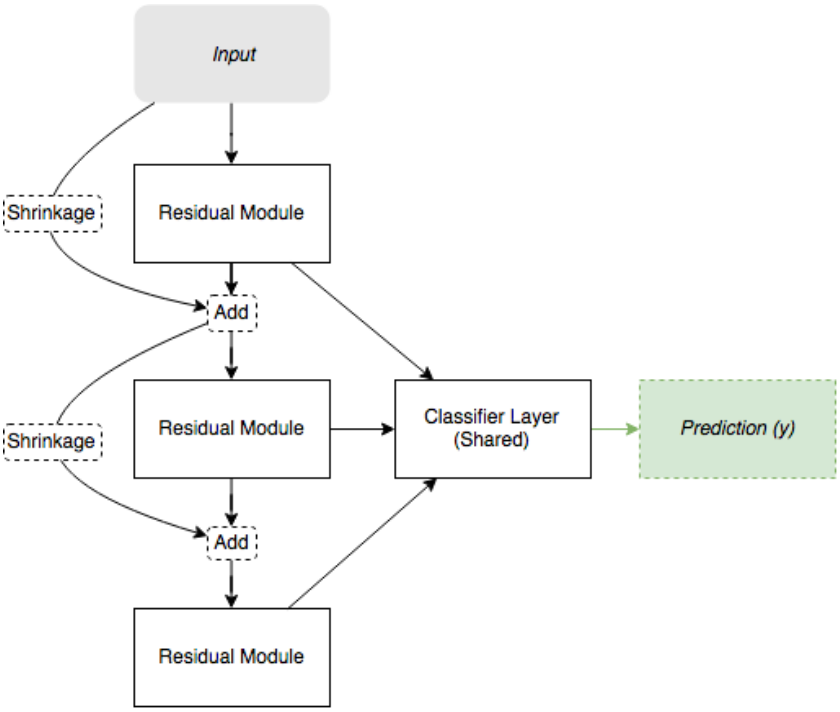


Figure 6.1: An example of a modified ResNet with shrinkage parameter using a shared linear layer.

constructs a different linear classifier layer which are dropped at the end of each sequential training phase. The usage of the combined auxiliary linear layer in our approach as a "shortcut" to the final layer is shown in Figure 6.1.

In our ResNet formulation, we can enable each module to all predict the same output $\mathbf{y}$ by allowing the final linear classifier layer to be shared with all the submodules, effectively acting as a shortcut. This formulation would have the output y_u^i and its corresponding weak learners $\mathbf{w}^\top \widehat{\mathcal{A}}^i$ product predictions as part of the neural network. This can be seen in lines 8, 9 of Algorithm 6.2. Hence each residual module would be updated with respect to the label $\mathbf{y}$.

It has been shown for ResNet that the paths exhibit a binomial distribution [69]. As such, the average path length is $\frac{n}{2}$ as there will be a path that moves across each module, and n routes that moves across a single module. Then even without leveraging a shared

Algorithm 6.2 ResNet with Shrinkage - Online Boosting Algorithm

Copy a given algorithm $\mathcal{A}$ N times, which we allow it to be: $\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^N$, with hyperparameter $\eta \in [\frac{1}{N}, 1]$ being the step size.
for $i = 1, 2, \ldots, N$ **do**
  set $\theta^i = 0$
end for
Define $\mathbf{F}_t^0 = 0$
for $u = 1, 2, \ldots, U$ **do**
▷ **Feed Forward**
  Sample $\mathbf{x}_u$
  for $i = 1, 2, \ldots, N$ **do**
    $\mathbf{F}_u^i = (1 - \theta^i)\mathbf{F}_u^{i-1} + \eta \mathbf{w}^\top \widehat{\mathcal{A}}^i(\mathbf{x}_u)$
    Calculate $\mathbf{y}_u^i = \psi^{-1}(\mathbf{F}_u^i)$
  end for
▷ **Back Prop**
  for $i = 1, 2, \ldots, N$ **do**
    Using back proagation, update all layers and shrinkage paramters in the sub-network up to module i using output $\mathbf{y}_u^i$,
  end for
  Evaluate algorithm using $\mathbf{y}_u^N$
end for

layer, the ResNet framework enables gradients to be recoved to all residual modules.

Through unravelling a ResNet, the paths of a ResNet are distributed in a binomial manner [69], that is, there is one path that passes through n modules and n paths that go through one module, with an average path length of $n/2$ [69]. This means that even without a shared layer, there will be paths within the ResNet framework where the gradient is recovered to the residual modules. This approach is shown by figure 6.2, and has an identical setup as algorithm 6.2 except in the back propagation step, we update all layers based on the whole network using output $\mathbf{y}_t^N$ only.

Remark: by using Algorithm 6.1, and reframing residual models as an Online Boosting we can see the equivalence when $\eta = 1$ in our hyperparamter selection, and if we fix the shrinkage parameter $\theta^i = 0$. Even with these adjustments, the regret bounds still so long as the residual modules themselves are bounded.

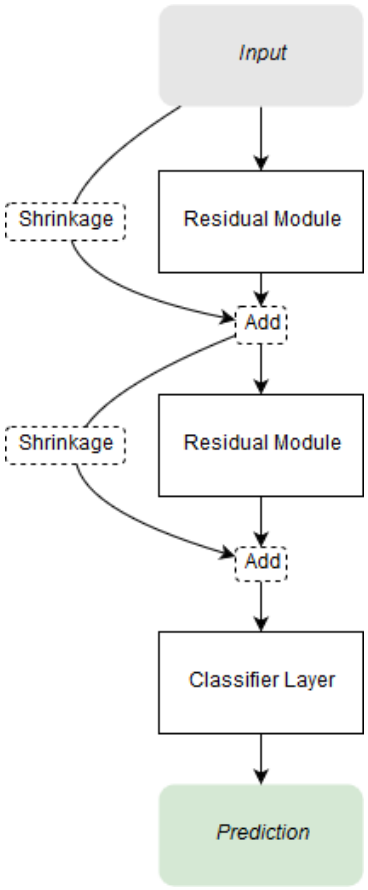


Figure 6.2: An example of the modified ResNet with two modules and shrinkage layers.

6.3.3 Constructing Decision Trees

In this section we explore how these can be constructed in our framework with online boosting guarantees and associated generalisation error analysis.

In this framework we reframe decision trees based on the approach taken by "Deep Neural Decision Forest" [39] as depicted in Figure 6.3. This formulation is equivalent to a feedforward neural network consisting of three layers. These three layers at a high level would be, a learnable layer representing the nodes in a tree, next is a predetermined layer used to determine the routing through the tree which is derived from the adjacency matrix when interpreting the decision tree as a graph structure, last layer is a learnable layer represents the leaf node output probabilities in the decision tree.

Furthermore, to ensure the appropriate regret bounds hold, the layers in the neural network representation must be bounded through l_p-norm, where $p \geq 1$, with activation functions being coordinate-wise and Lipschitz constant of 1. To determine the size of each layer, this is reflective of the number of nodes in the decision tree structure which is pre-determined before training. Let the decision tree nodes to be a pre-selected number n, with the input size of the feature vector to the decision tree module to be size k. Next we describe the construction of each layer.

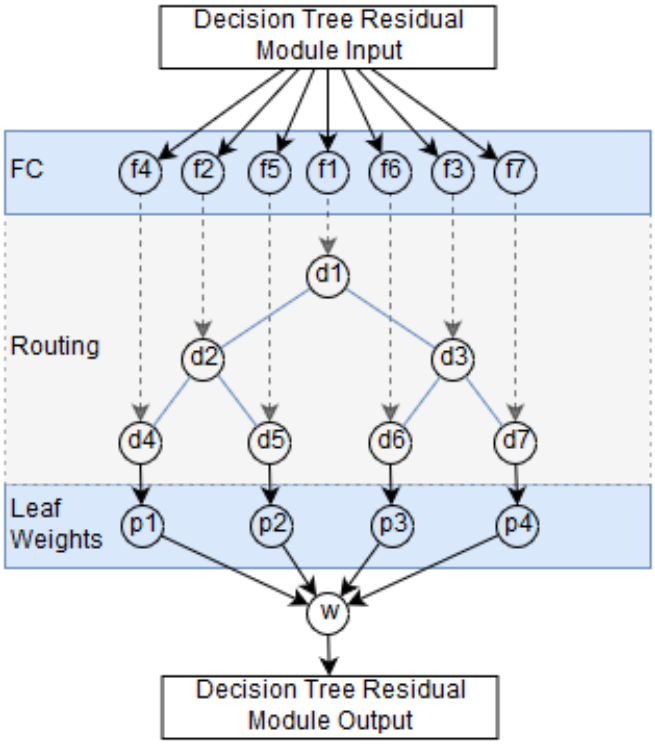


Figure 6.3: ResNet module based on "Deep Neural Decision Forests" [39]. This forms the decision tree ResNet module.

We begin with the input of the decision tree module, which determines the route taken by each node. The decision splits in this setting are typically oblique decision splits. This can be determined by using learnable parameters $\Theta = \{W, b\}$, where $W \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^n$. Then the output would be $H_1(x) = \widehat{W}^\top x + \widehat{b}$ with x being the input to the module,

where $\widehat{W} = [+W \oplus -W]$ and $\widehat{b} = [+b \oplus -b]$, and $\oplus$ is the concatenation operation.

The second layer is predetermined and based on the structure of the tree provided. It is based on the adjacency matrix when representing a tree as a graph which informs the decision tree neural network structure which nodes would route to the corresponding leaf nodes. We introduce this through a matrix $Q \in \{0,1\}^{2n \times (n+1)}$. To ensure the 1-Lipschitz bounded function condition holds, let $\phi_2(x) = \log \circ \text{softmax})(x)$ be activation function where $\circ$ represents function composition. As $\log(x)$ is 1-Lipschitz is bounded and in the domain $(0,1)$ and in the range $\text{softmax}(x) \in (0,1), \forall x$, then this is an allowable activation function, with $H_2(x) = Q^\top(\phi_2 \circ H_1)(x)$ to be the second layer output.

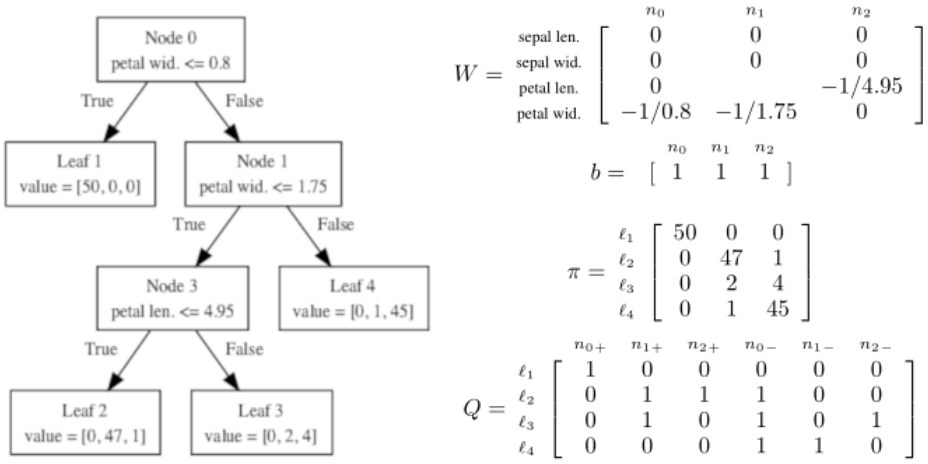


Figure 6.4: Left: Scikit-Learn Decision Tree, Right: Neural Network Decision Tree. Using a hard max function will generate the appropriate decision tree.

The final layer is the probability output of the leaf layer in the neural network. This is a learnable linear layer based on the leaf layer, parameterised by $\pi \in \mathbb{R}^{n+1}$. Then $\pi^\top(\phi_3 \circ H_2(x))$ is the output of the decision tree with activation function $\phi_3(s) = \exp(x)$. As the range of $H_2(x)$ is $(-\infty, 0)$, consequently as the domain of $\exp(x)$ is $(-\infty, 0)$ then $\phi_3(x)$ is a 1-Lipschitz bounded function. As the construction of our neural network is a series of feedforward layers, then it is part in the artificial neural networks family as AdaNet, leading to the same generalisation bounds. The summary of the three layer feedforward neural network is shown in Figure 6.4. To extend this approach to ResNet,

the feature representation must be invariant to output size of the ResNet module. To accomplish this a linear layer is added to guarantee that the shortcut has precisely the same dimensions as required for compatibility with the decision tree modules.

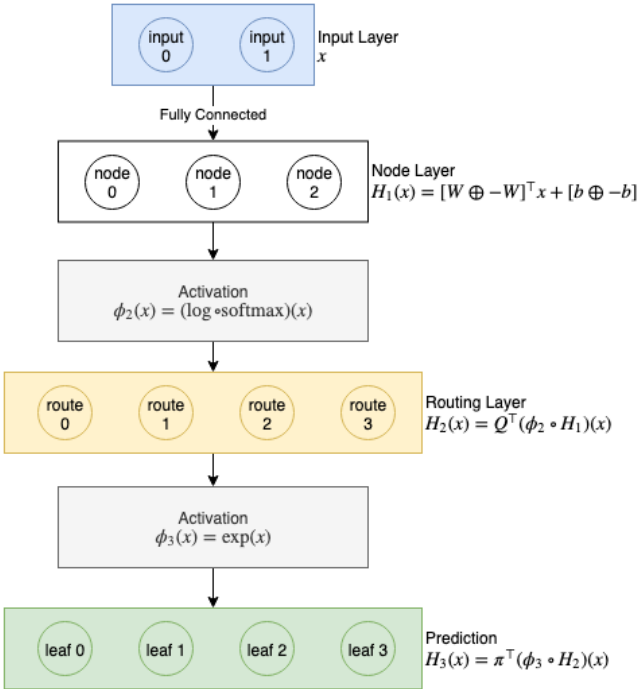


Figure 6.5: A decision tree can be represented by neural networks using three layers. This representation is constructed with a trainable decision tree and leaf nodes.

The formulation of these equations and their parameters is shown in figure 6.4 which demonstrates how a decision tree trained in Python Scikit-Learn can have its parameters be converted to a decision tree in our neural network formulation, and figure 6.5 demonstrates the formulation of the three layer network which constructs this decision tree.

6.4 Experiments

For our experiments, we explore two different modifications to ResNet architectures. In the first setting, we examine modifying ResNet with convolution modules that are used

for image classification and in the second instance we modify ResNet to be used in the tabular setting by using decision tree modules.

We first explore image datasets in which we use the convolution network variation and examine the impact when adding trainable shrinkage parameter and dense linear layer over image classification tasks.

In the tabular setting, we compare the boosted decision models with ResNet decision tree modules. This was compared with other approaches both in the neural network setting [39] and the offline setting [37] with the results shown in table 6.3 where we conduct our training and testing datasets with 70%, 30% split.

We also apply our approach to reinforcement learning which is conducted over a range of benchmark reinforcement learning and multiagent reinforcement learning benchmark tasks.

6.4.1 Image Recognition Results

For the image recognition benchmark we use exactly the same 20-layer ResNet as a point of comparison, as used in the official implementation. To generate our variation, we add a learnable shrinkage parameter (*ResNet-Shrinkage*), and also a shared linear layer (*ResNet-Linear Projection*).

MODEL	TEST	TRAIN
RESNET	0.94630	0.93886
RESNET (W/ SHRINKAGE)	**0.94852**	**0.93917**
RESNET (LINEAR PROJECTION)	0.94626	0.93689

Table 6.1: Performance of the "Street View House Number" Task. The modules below leveraged similar hyperparameters, for example all models are standardised with number of iterations and the learning schedule of the Optimisers.)

MODEL	TEST	TRAIN
RESNET	0.91530	0.98412
RESNET (W/ SHRINKAGE)	**0.91870**	**0.98504**
RESNET (LINEAR PROJECTION)	0.88570	0.94496

Table 6.2: Performances of the CIFAR-10 Task. The modules below leveraged similar hyperparameters, for example all models are standardised with number of iterations and the learning schedule of the Optimisers.)

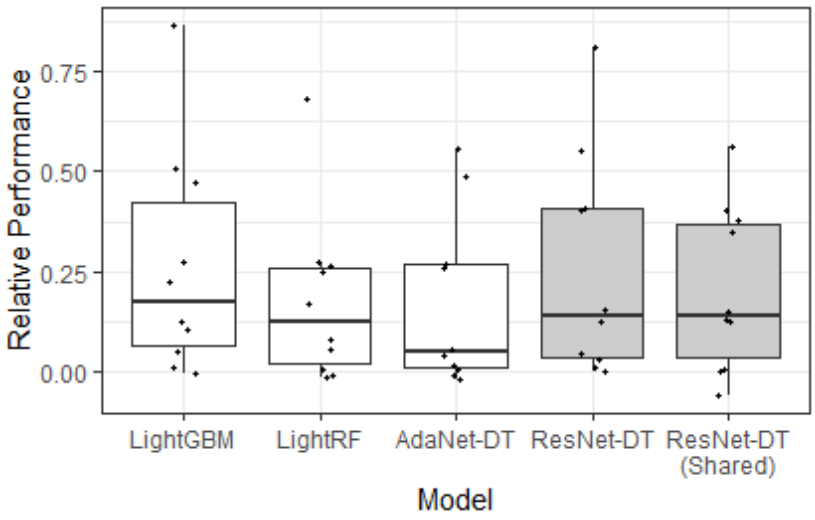
From the results, using shrinkage outperforms the vanilla ResNet model in both datasets, however the model with shared linear appears to perform worse in the CIFAR-10 challenge. Based on our experiments, the addition of a shrinkage parameter can be included to improve the performance of ResNet models.

6.4.2 Gradient Boosting Decision Tree

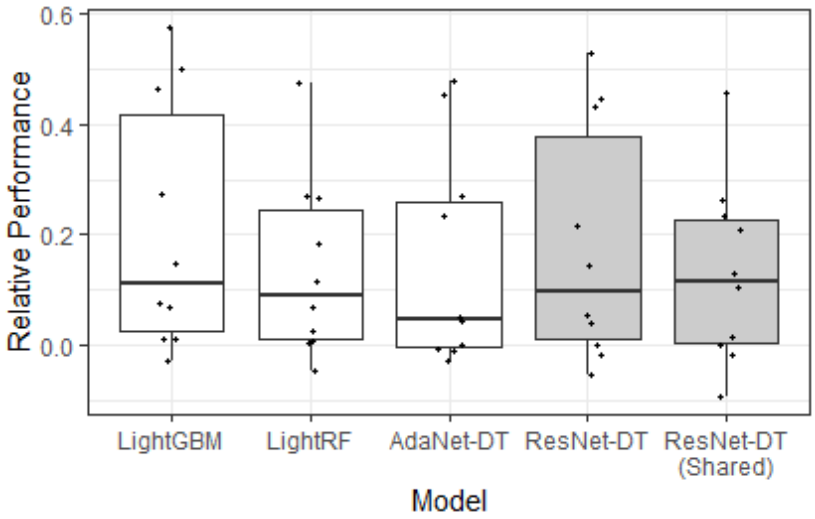
For comparison in the tabular scenario, we examine the ability to use different residual module representation to construct a variety of models based on ResNet. Using residual network modules, decision trees can be constructed and trained as a ResNet (ResNet-DT), and also its shared linear representation (ResNet-DT Linear Projection), and using AdaNet to with decision trees (AdaNet-DT). As a comparison we used LightGBM for their boosted decision trees (Light-GBDT) and random (LightRF) forest algorithms, and used *Deep Neural Decision Forests* (DNDF) as the baseline approach. To ensure the experiments are comparable, we keep the hyperparameters for all models consistent, with each model having a maximum of 15 trees, depth of 5. For all neural network approaches, the models were trained for 200 epochs.

Dataset	ResNet-DT		ResNet-DT (Linear Proj.)		DNDF		AdaNet-DT		LightRF		LightGBM	
	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train
adult	0.8386	0.8837	0.7733	0.8057	0.8538	0.8579	0.8533	0.8628	0.8596	0.8624	0.8613	0.8638
covtype	0.8302	0.9689	0.8246	0.9457	0.6825	0.6877	0.7106	0.7153	0.7598	0.8030	0.7831	0.8396
dna	0.8848	0.9888	0.9204	0.9830	0.9361	0.9794	0.9288	0.9919	0.9393	0.9632	0.9466	0.9745
glass	0.6719	0.7933	0.5781	0.7933	0.4688	0.5667	0.5781	0.7133	0.5938	0.7067	0.7031	0.8533
letter	0.9700	0.9934	0.9590	0.9868	0.8495	0.8602	0.8383	0.8524	0.8693	0.9056	0.9075	0.9481
sat	0.8710	0.8904	0.9125	0.9628	0.8275	0.8519	0.8695	0.8976	0.8835	0.9204	0.8885	0.9556
shuttle	0.7859	0.7860	0.7859	0.7860	0.7859	0.7860	0.9964	0.9965	0.9985	0.9992	0.9997	0.9997
mandelon	0.9078	0.9848	0.8856	0.9881	0.8722	0.8776	0.8478	0.8576	0.8322	0.8700	0.8456	0.9219
soybean	0.8873	0.9916	0.8922	0.9979	0.6127	0.6388	0.9069	0.9937	0.7255	0.8058	0.8971	0.9415
yeast	0.5910	0.7382	0.4876	0.5486	0.3865	0.4071	0.5618	0.6064	0.5708	0.6843	0.6090	0.7594
Number of Top Scores	3	3	1	3	0	0	1	1	0	0	5	3
Mean Reciprocal Rank	0.5117	0.5783	0.4067	0.5450	0.2283	0.1983	0.3150	0.345	0.3367	0.2683	0.6700	0.5450

Table 6.3: Performance of the decision tree module across a variety of supervised learning benchmark datasets.



(a) Relative performance over the training dataset.



(b) Relative performance over the test dataset.

Figure 6.6: he relative performance across a range of algorithms. We use "Deep Neural Decision Forest" as the baseline model. Higher scores indicate superior performance.

To ensure comparability among all models, we considered the median and mean error improvement compared with a baseline model. The baseline model chosen was the DNDF model as they were the overall worst performing model in our benchmarks. Whilst *LightGBM* appeared to be the best average improvement compared with the baseline, both of our approaches *ResNet-DT* performed second best, being superior to

	MEAN IMPROVEMENT	MEAN RECIPROCAL RANK
LIGHTGBM	**26.140%**	*0.545*
LIGHTRF	17.414%	0.2683
ADANET-DT	16.519%	0.345
RESNET-DT	*25.360%*	**0.5783**
RESNET-DT (LINEAR PROJECTION)	20.306%	*0.545*

Table 6.4: The relative performance across a range of algorithms. We use "Deep Neural Decision Forest" as the baseline model. We calculate Mean Reciprocal Rank on the training splits.

	MEAN IMPROVEMENT	MEAN RECIPROCAL RANK
LIGHTGBM	**20.904%**	**0.67**
LIGHTRF	13.665%	0.3367
ADANET-DT	14.771%	0.315
RESNET-DT	*17.892%*	*0.5117*
RESNET-DT (LINEAR PROJECTION)	12.949%	0.4067

Table 6.5: The relative performance across a range of algorithms. We use "Deep Neural Decision Forest" as the baseline model. We calculate Mean Reciprocal Rank on the testing splits.

LightGBM's Random Forrest variation and *AdaNet-DT*. In our experiments, *AdaNet-DT* was restricted from building "deep" models, and were only have to construct "bushy" models compared with *ResNet-DT*. Even with these restrictions, the *AdaNet-DT* approach outperformed the baseline implementation.

To construct a baseline for all models to be comparable, the results presented are on the average and median error improvement compared with *DNDF* models, as they were the worse performing model based on these benchmarks. From the results in Tables 6.4 and 6.5, *LightGBM* performed the best with the best average improvement on error relative to the baseline DNDF model. What is interesting is that both our *ResNet-DT* model performed second best, beating *LightRF* and *AdaNet-DT* models. It is important to note that our setup for *AdaNet-DT* only allowed a "bushy" candidate model, this

did not allow *AdaNet-DT* to build deeper layers compared with *ResNet-DT* approach; only allowing it to build a wider and shallow architect through appending additional decision trees. Despite this, the *AdaNet-DT* implementation did outperform the *DNDF* implementation.

To understand the overall performance, rather than looking at single statistics, it is important to understand how the model performance is distributed overall. This is shown in Figure 6.6 which demonstrate the distribution of the train and test datasets. Based on these results, we observe *ResNet-DT* and *RestNet-DT Linear Projection* have similar variations in their results.

Overall the performance between *ResNet-DT* and *LightGBM* is neglible with our approach providing additional flexibility to update the tree ensemble via minibatch and the ability to produce non-greedy decision rules. We can use online learning to enable training over large datasets through incremental minibatch updates, to allow adaptations for data drift, whereas *LightGBM* is unable to operate in this manner.

6.5 Reinforcement Learning Experimental Results

The ResNet module approach can be extended beyond supervised learning. Recently there has been success in re-interpreting Residual Networks to learn underlying feature representation in reinforcement learning space [61]. In this section we apply ablation studies for our ResNet modules using Q-Learning approaches in the MARL space. In this setting rather than interpreting each module as a classifier, we can instead reframe the problem to suggest that each module is an independent agent where during training time, the Q values can be combined in a similar way as the shared linear classifier module.

We first examine our approach as a drop-in replacement to DQN models where the MLP are replaced with a model with skip connections and shrinkage. The ablation study

is conducted over a range of benchmark reinforcement environments built on top of DQN. The ablation environments use DQN as the baseline model, DQN with ResNet modules (DQN-RN), DQN with ResNet module with shrinkage (DQN-RN+). Our environment setup uses the default settings, except for GridWorld [8] environments which include using the inbuilt *Action Bonus* and *FlatObs* wrappers. The task in GridWorld is to navigate a grid to fetch an item given a prompt, where the actions and movement in the space are purely discrete. In GridWorld, an episode is terminated upon timeout or when an agent picks up an object irrespective whether it is the correct one or not. A reward is provided for successfully picking up the correct object, whereas a negative reward is provided if the incorrect object is retrieved. Acrobot is a classic control environment where the goal is to apply torque on the actuated joint to swing the free end of a line chain above a given height. Finally Lunar Landing involves optimizing a rocket's trajectory path to determine when it is optimal to fire the engine at full throttle or turn it off. The act of turning off an engine along with directionality means this environment is also a discrete action environment.

The results from our RL experiments can be seen from Figure 6.7 where DQN-RN approaches are comparable and in general superior to DQN approaches with DQN-RN+ edging out over DQN-RN approaches.

Next we evaluate our ResNet approach in the context of MARL environments with the following variations: IQL model as a baseline (IQL), IQL with ResNet modules (IQL-RN), IQL with ResNet modules with shrinkage (IQL-RN+), finally QMIX style module where the mixing network is the sum of the sub-agents Q values with shrinkage (QMIX-RN).

To ensure consistency across tasks and environments, consistent preprocessing was used as suggested in other MARL benchmark [68]. When evaluating our agents, the tasks were run in separate evaluation environments with 1000 episode roll outs. From

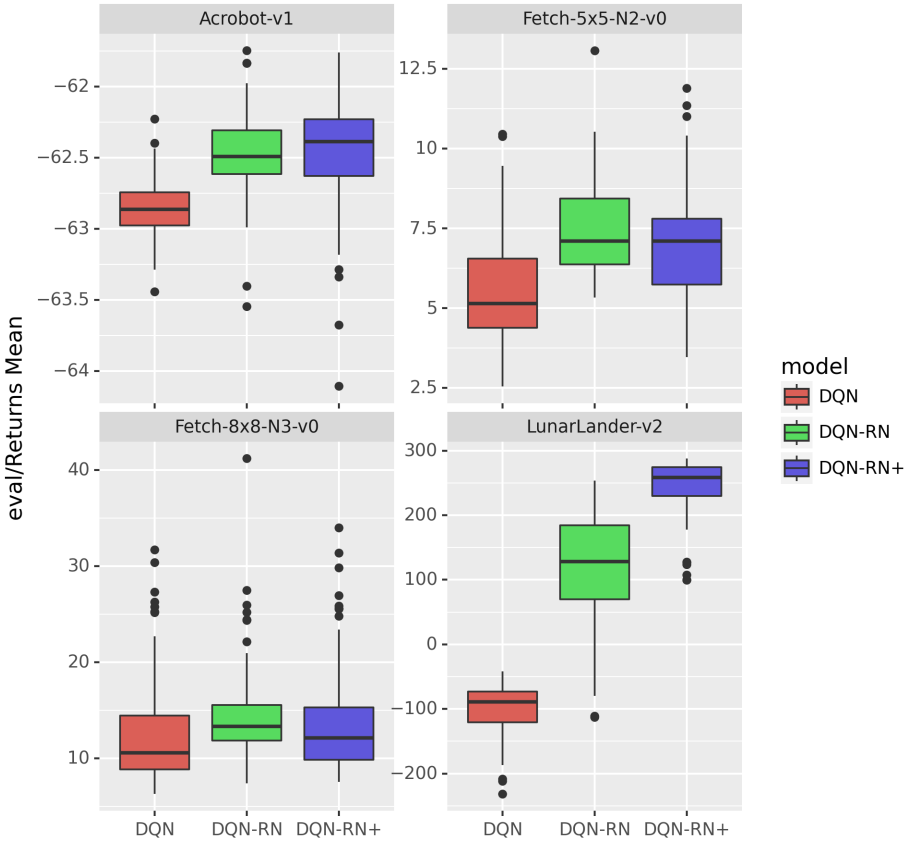


Figure 6.7: The returns from benchmark Reinforcement Learning problems. The return is provided over the last ten epochs. Higher scores indicate superior performance.

the results shown in Table 6.6, we first notice that simply replacing the standard MLP with ResNet modules does not guarantee improved performance, but in fact often leads to worse performance! On the contrary, by having a shrinkage parameter we yield consistently better results than IQL. Consistent with the results from the supervised learning tasks, we can also see the addition of the additive shrinkage becomes a hyperparameter choice similar to the comparison between the decision to use a shared module in the supervised scenario.

	IQL	IQL-RN	IQL-RN+	QMIX-RN
pursuit	11.79 ± 6.64	11.89 ± 6.78	12.98 ± 6.74	12.25 ± 6.80
reference	1.06 ± 0.52	1.042 ± 0.55	1.18 ± 0.60	1.65 ± 0.82
spread	1.58 ± 0.81	1.62 ± 0.86	1.78 ± 0.93	2.03 ± 1.04
tag	43.80 ± 1.09	25.2 ± 1.077	27.6 ± 1.29	7.8 ± 1.30
waterworld	6.44 ± 3.48	6.29 ± 3.45	7.49 ± 3.72	7.52 ± 3.91

Table 6.6: Final Evaluation returns with one standard deviation for multi-agent environments using ResNet modules as a drop-in replacement for "Independent Q-learning" for MARL tasks using PettingZoo environments.

6.6 Conclusions

We end this chapter by demonstrating the relationship between ResNet and Boosting through interpreting the residual modules as linear classifier of weak learners. By adding a shrinkage parameter, it has provided promising results in refining ResNet models. By extending this approach to decision trees, our approach also enables learning decision tree ensembles without the downsides of offline approaches. These insights into the ResNets and Boosting could be used to spur other novel networks and provide various interpretation and justification into neural network architecture choices. These choices and subsequent reinterpretation of ensemble networks can be applied to MARL environments as a special kind of mixer networks in a QMIX style model which we demonstrated can yield superior performance to MLP with minimal changes.

7

Greedy UnMixing for Multi-Agent Reinforcement Learning

Contents

7.1 Introduction

In this chapter we continue our journey in examining the challenge of *collaboration* in MARL tasks. In the previous chapters we considered models part of the Centralised Training with Decentralised Execution (CTDE) framework which leveraged centralised structures. This raises a key question - what if this centralised structure is redundant or even harmful to our learning process? Could we "unmix" this representation, and thereby remove this altogether? Greedy UnMix (GUM) is our approach which aims to achieve this through treating the decision for having the centralised mixing structure as a neural architecture choice. In doing so, GUM enables the ability to simplify the problem space, and ensembles the individual agent state action functions via a distributional RL approach.

Through our empirical experiments we demonstrate various ways in how we can combine our agent state action functions with a mixer network and decouple them under the lens of ensemble learning. We demonstrate the importance of Q-ensemble choices by exploring different approaches and justifying our decision in thorough ablation studies. GUM is also evaluated across a range of MARL benchmark environments in which we demonstrate comparable results with existing approaches despite its relative simplicity.

7.2 Background

Here we provide an overview of key concepts which are important to this chapter. This thesis is focussed on the decentralised partially observable setting, defined by tuple

$\langle S, U, P, r, Z, O, n, \gamma \rangle$, with $s_t \in S$ being the environments state, and t is the corresponding timestep, with $t = 1, 2, \ldots$ [53]. At each timestep every agent $a \in A \equiv \{1, \ldots, n\}$, selects action $u \in U$, with $\mathbf{u} \in \mathbf{U} \equiv U^n$ being defined as the joint action space. Additionally, the we define $s'inS$ to be the next state. This is drawn according to $P(\cdot|s, \mathbf{u})$ which is the state transition function. We are interested in cooperative RL where all agents share the same reward, so the reward for the whole environment is $r(s, \mathbf{u})$, the discount factor $\gamma \in (0, 1)$. In this these, the partial observable setting is considered. This is denoted to be $z \in Z$ which is generated by $O(s, i) : S \times A \rightarrow Z$. Furthermore, for a particular agent a, we define τ to be $\tau^a \in T \equiv (Z \times U)^*$, which is agent a's action observation history; on which it conditions its stochastic policy $\pi^i(u^i|\tau^i) : T \times U \rightarrow [0, 1]$. Where π is the joint policy which is the Q function or state action function: $Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[R_t|s_t, \mathbf{u}_t]$, with the *discounted reward* defined as $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$.

7.2.1 Centralised Training Decentralised Execution

In this chapter, we again focus on the Centralised Training Decentralised Execution (CTDE) framework, where training occurs in a centralised fashion, whilst during evaluation, the execution of the agents is decentralised. More specifically, during training phase, the model can access global state information and any agent's action-observation history. During evaluation, agents only act according to their own τ^i which is the decentralised execution.

7.2.2 Learning from Suboptimal Experiences

One recent area of interest in reinforcement learning (RL) is the focus in learning and constraining policy updates when only suboptimal data is available. This is often framed in the context of *offline* RL, whereby often the key to resolving this challenge is recognising that there are temporal differences when bootstrapping against a target network [41].

This often leads to overly optimistic Q function estimates particularly when estimating actions that are unseen or out of distribution. This necessitates algorithms which go beyond imitation learning, as imitation learning typically presumes the existence of optimal data. Solutions to resolve this include avoiding out-of-distribution actions based on the behaviour policy distribution [20, 41, 72], or using a distributional reinforcement learning approach which can attempt to reduce the overestimation by providing lower-bound value estimates [19, 42].

7.2.3 Multi-Agent Reinforcement Learning

In this chapter, we focus on MARL approaches part of the CTDE framework. Perhaps the most straight-forward approach is Independent Q-Learning (IQL) [67]. In IQL, there is no centralised training (or more specifically, there is no additional state information used), and the agents will train and execute independently from each other, based only on their local observation. To increase the training speed, parameter sharing is typically used. To combine Q state action functions, various approaches are used which can leverage centralised training structures. For example VDN [64] creates a centralised structure through the summation of each agent's independent state action function, QMIX [57] uses monotonic neural network hypernetwork as a centralised structure to combine independent Q functions and QTRAN [63] allows the usage of any arbitrary function by modifying the objective to ensure effective computation of the argmax in the state action function. Policy gradient and actor critic methods can also be used in this framework. For example MADDPG [45] leverages independent actors for decentralised execution, but allows the critics to have access to other agent's state, whereas LICA [76] leverages a centralised critic with an adaptive entropy constraint to ensure consistent exploration by all agents when training.

7.3 Methodology

To describe our methodology we first provide some preliminary background covering how we extend independent Q learning to the distributional RL approach and extending it to an ensemble approach to enable unmixing the joint Q-learning algorithm.

7.3.1 Conservative Q-Learning in Multi-Agent Settings

Next, we examine the considerations required when adaptive Conservative Q-Learning to MARL setting. Before that, we provide a brief overview of Conservative Q-Learning (CQL) [42]. Conservative Q-Learning is an adaptation of distributional RL approach in the DQN space. The distributional RL approach that is used is the Quantile Regression DQN [12]. The changes made to DQN arem we first convert the output to emit quantile estimates, and then modify the corresponding loss estimate from mean squared error to quantile Huber loss. The additional change which CQL makes is the inclusion of the regulariser penalty which minimises the expectation of the Q state action function based on a behaviour policy μ, which is constructed to match the state-marginal based on the experiences, which we denote by $\mathbb{E}_{\tau\sim\mathcal{D},u\sim\mu}[Q(\tau,u)]$. Let $\hat{\beta}(u|\tau) := \frac{\sum_{\tau,u\in\mathcal{D}}\mathbf{1}[\tau=\tau,u=u]}{\sum_{\tau\in\mathcal{D}}\mathbf{1}[u=u]}$, where $\mathcal{D}$ is our dataset of experiences denote the empirical behaviour policy. This leads to the iterative update for CQL algorithm.

In the independent setting, as each agent has their own Q state action function, then each agent would "conservatively" estimate Q, allowing theoretical guarantees in CQL to hold. Given that we operate in CTDE framework, the next question would be to ask under what conditions and which appropriate Q value factorisation would the CQL guarantees still hold?

As the QMIX centralised mixing function leverages monotonic transformations using $Q^1, Q^2 \ldots Q^n$, which is the Q state action function of each agent. The naive solution is to regularise with respect to the whole hypernetwork, however this approach is not required!

This is because regularising the individual agent Q state action functions (i.e. agent policies) is sufficient to provide a lower bound on the algorithm execution. This means we do not necessarily need to regularise or interact with the mixing hypernetnetwork. We will explore the effects of both scenarios in the ablation studies. As the addition to the setup is only the regulariser term, then the independent global max formulation [63] would still hold.

7.3.2 Greedy UnMixing

The Greedy UnMixing algorithm (GUM) is applicable to any MARL Q-learning algorithm with a centralised mixing network such as algorithms belonging to the QMIX family including QMIX, VDN, QTRAN, QCGraph. The key insight is the ability to *control* the influence of this centralised mixing network, treating this decision to be a neural architecture selection problem. To induce this, we leverage a learnable parameter that controls the mixing strength between the independent and centralised networks. This weight can be interpreted as a trade-off parameter, whereby the larger the weight, the greater the influence of the central mixing network. As GUM learns, the trade-off parameter may learn to be close to zero, in which case our algorithm would revert to the independent Q-learning framework and in essence remove the centralised mixing components.

The CQL framework extends the iterative update step in Q-learning as follows:

$$\hat{Q}^{\pi}_{\text{CQL}} \leftarrow \underset{Q}{\text{argmin}} \underbrace{\alpha \cdot \left(\mathbb{E}_{\tau\sim\mathcal{D},u\sim\mu}[Q(\tau,u)] - \mathbb{E}_{\tau\sim\mathcal{D},u\sim\hat{\beta}}[Q(\tau,u)]\right)}_{\text{(off-policy) CQL regulariser}} + \frac{1}{2}\underbrace{\mathbb{E}_{\tau,u,\tau'\sim\mathcal{D}}\left[(r(\tau,u)+\gamma\mathbb{E}_{\pi}[\bar{Q}(\tau',u')] - Q(\tau,u))^2\right]}_{\text{standard TD error}} \tag{7.1}$$

Then in the GUM setup, the regulariser term from Equation 7.1 is denoted as $\mathcal{R}$, is used for the mixer network in the MARL setting and denoated as $\mathcal{R}_{\text{mixer}}$. Our

learning parameter is represented as α and α_{mixer} respectively and the neural network architecture is shown in Figure 7.1. The update equation is shown below

$$\hat{Q}^{\pi}_{\text{GUM}} \leftarrow \alpha\mathcal{R} + \alpha_{\text{mixer}}\mathcal{R}_{\text{mixer}}$$
$$+\frac{1}{2}\mathbb{E}_{s,\mathbf{u},s'\sim\mathcal{D}}\left[(r(s,\mathbf{u}) + \gamma\mathbb{E}_{\pi}[\bar{Q}_{\text{mixer}}(s',\mathbf{u}')] - Q_{\text{mixer}}(s,\mathbf{u}))^2\right] \tag{7.2}$$

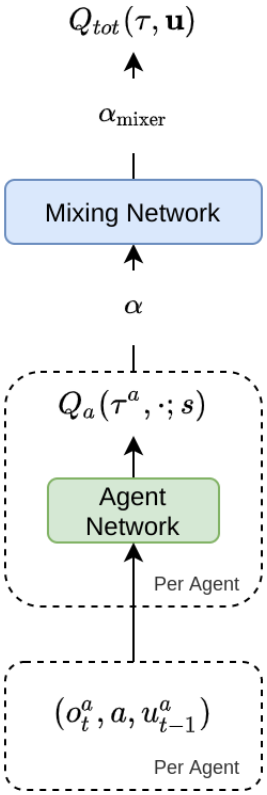


Figure 7.1: GUM Algorithm Architecture. In the diagram above, a is used to indicate the agent index, whereas α and α_{mixer} indicates the trade-off parameters. We observe that the model is first created without the mixer network, and is cloned over with a learned trade-off parameters α and α_{mixer}. Then the neural network can perform neural architecture search decision and assign weights to the independent agent policies and the mixer network to avoid overfitting.

To conclude, we have laid out the architect of GUM, and described how we can implicitly unmix the independent and mixing networks through leveraging regulariser

strength. In the next section, we reinterpret GUM in the ensemble learning framework, and position GUM as a neural network architecture problem in the AdaNet framework.

7.3.3 Greedy UnMix as an Ensemble

The neural network architecture selection problem has been explored through AdaNet framework which proposes candidate modules with hard selections [11], or through soft selection [62]. The GUM algorithm can be reformed to leverage soft proposals using a trade-off parameter α. Under this approach the neural network architecture selection reforms to having the base architecture to be the independent Q-learning network with a soft proposal involving the central mixing network. This is not the only approach to ensemble Q learning functions, and has been explored elsewhere in other approaches for gaining better estimations of the value functions, typically in the actor-critic methods [20, 41, 44]. These variations are explored in the ablation sections of the experiments. The final output of the GUM model under the explicit approach framed as a neural network architecture selection problem is then to leverage the monotonic function factorisation $Q_{\text{adanet}} = \alpha[Q_1, \ldots, Q_n] + \alpha_{\text{mixer}} Q_{\text{mixer}}$, with learnable parameters α and α_{mixer} using gradient ascent.

$$\hat{Q}^{\pi}_{\text{GUM-adanet}} \leftarrow \mathbb{E}_{s,\mathbf{u},s' \sim \mathcal{D}} \left[(r(s,\mathbf{u}) + \gamma \mathbb{E}_{\pi}[\bar{Q}_{\text{adanet}}(s',\mathbf{u}')] - Q_{\text{adanet}}(s,\mathbf{u}))^2 \right] \tag{7.3}$$

Algorithm 7.1 Greedy UnMixing

1: For all agents $\{Q^i\}_{i=1}^n$, set the Q state action and the mixer hypernetwork Q_{mixer}, trade off factor $\alpha, \alpha_{\text{mixer}}$.
2: **for** timestep $t = 1, \ldots, N$ **do**
3: Update $\{Q^i\}_{i=1}^n$ and the mixing network, Q_{mixer} functions, via objective from Equation 7.2 or Equation 7.3 for the implicit or explicit approach respectively.
4: For the trade off factors $\alpha, \alpha_{\text{mixer}}$.
5: **end for**
6: Use decentralised execution to evaluate via $\{Q^i\}_{i=1}^n$

To conclude, we have presented the two variations (implicit and explicit unmixing) related to GUM.

7.3.3.1 Analysis of GUM Algorithm

The formulation of the GUM algorithm is based on a multi-agent extension of Conservative Q-Learning.

Theorem 7.1. (Lower bound estimate of the true policy is obtained through Equation 7.2) *Under the Q_{GUM} function from Equation 7.2 the value of an independent agent's policy, $\hat{V}^{\pi}(s) = \mathbb{E}_{\pi}[\hat{Q}_{GUM}(s,u)]$, leads to the generation of a lower bound to $V^{\pi}(s) = \mathbb{E}_{\pi}[Q_{GUM}(s,u)]$ which is the true value function, with $\mu = \pi$ for any $\alpha > 0$.*

Proof. We can compute the Q function update as per the objective in Equation 7.2, with k being the policy update iteration. Before determining the lower bound estimate of the true policy, we first consider value function update below:

$$\begin{aligned}\hat{V}_{k+1}(\tau) &:= \mathbb{E}_{\mathbf{u}\sim\{\pi_i(u|\tau)\}_{i=1}^{n}}\left[\hat{Q}_{k+1}(s,\mathbf{u})\right] \\ &= \mathcal{B}^{*}\hat{V}^{k}(\tau) - \alpha\left(\sum_{i=1}^{n}\mathbb{E}_{u\sim\pi_i(u|\tau_i)}\left[\frac{\mu_i(u|\tau_i)}{\beta(u|\tau_i)} - 1\right]\right) \\ &\quad - \alpha_{\text{mixer}}\mathbb{E}_{\mathbf{u}\sim\pi_{\text{mixer}}(\mathbf{u}|s)}\left[\frac{\mu_{\text{mixer}}(\mathbf{u}|s)}{\beta(\mathbf{u}|s)} - 1\right]\end{aligned}$$

For clarity in notation, we drop the i, then for any particular agent, if $\mu = \pi$, $\mathbb{E}_{u\sim\pi(u|\tau)}\left[\frac{\mu(u|\tau)}{\beta(u|\tau)} - 1\right]$. Following this result:

$$\begin{aligned}
\mathbb{E}_{u\sim\pi(u|\tau)}\left[\hat{Q}_{k+1}(\tau,u)\right] &= \sum_u \pi(u|\tau)\left[\frac{\mu(u|\tau)}{\beta(u|\tau)}-1\right] \\
&= \sum_u (\pi(u|\tau)-\beta(u|\tau)+\beta(u|\tau))\left[\frac{\mu(u|\tau)}{\beta(u|\tau)}-1\right] \\
&= \sum_u (\pi(u|\tau)-\beta(u|\tau))\left[\frac{\pi(u|\tau)-\beta(u|\tau)}{\beta(u|\tau)}\right] \\
&\quad + \sum_u \beta(u|\tau)\left[\frac{\mu(u|\tau)}{\beta(u|\tau)}-1\right] \\
&= \sum_u \left[\frac{(\pi(u|\tau)-\beta(u|\tau))^2}{\beta(u|\tau)}\right] + \sum_u \mu(u|\tau) - \sum_u \beta(u|\tau) \\
&= \sum_u \left[\frac{(\pi(u|\tau)-\beta(u|\tau))^2}{\beta(u|\tau)}\right] + 0 \\
&\geq 0
\end{aligned}$$

Then $\sum_{i=1}^{n} \mathbb{E}_{u\sim\pi_i(u|\tau_i)}\left[\frac{\mu_i(u|\tau_i)}{\beta(u|\tau_i)}-1\right]$ is non-negative, with each agent regulariser being independent and additive. A similar approach is used when examining the joint state action mixer component $\mathbb{E}_{\mathbf{u}\sim\pi_{\text{mixer}}(\mathbf{u}|s)}\left[\frac{\mu_{\text{mixer}}(\mathbf{u}|s)}{\beta(\mathbf{u}|s)}-1\right]$.

$$\begin{aligned}
\mathbb{E}_{\mathbf{u}\sim\pi_{\text{mixer}}(\mathbf{u}|s)}\left[\frac{\mu_{\text{mixer}}(\mathbf{u}|s)}{\beta(\mathbf{u}|s)}-1\right] &= \sum_{\mathbf{u}\in\mathbf{U}} \pi_{\text{mixer}}(\mathbf{u}|s)\left[\frac{\mu_{\text{mixer}}(\mathbf{u}|s)}{\beta(\mathbf{u}|s)}-1\right] \\
&= \sum_{\mathbf{u}\in\mathbf{U}} \Big(\left(\pi_{\text{mixer}}(\mathbf{u}|s)-\beta(\mathbf{u}|s)+\beta(\mathbf{u}|s)\right) \\
&\quad \times \left[\frac{\mu_{\text{mixer}}(\mathbf{u}|s)}{\beta(\mathbf{u}|s)}-1\right]\Big) \\
&= \sum_{\mathbf{u}\in\mathbf{U}} \left[\frac{(\pi_{\text{mixer}}(\mathbf{u}|s)-\beta(\mathbf{u}|s))^2}{\beta(\mathbf{u}|s)}\right] \\
&\quad + \sum_{\mathbf{u}\in\mathbf{U}} \mu_{\text{mixer}}(\mathbf{u}|s) + \sum_{\mathbf{u}\in\mathbf{U}} \beta(\mathbf{u}|s) \\
&\geq 0
\end{aligned}$$

We observe that $\mathbb{E}_{u\sim\pi_i(u|\tau_i)}\left[\frac{\mu_i(u|\tau_i)}{\beta(u|\tau_i)}-1\right], \forall i$ and $\mathbb{E}_{\mathbf{u}\sim\pi_{\text{mixer}}(\mathbf{u}|s)}\left[\frac{\mu_{\text{mixer}}(\mathbf{u}|s)}{\beta(\mathbf{u}|s)}-1\right]$ is non-negative. Then when considering the GUM algorithm, each value iterate induces underestimation $\hat{V}_{k+1}(s) \leq \mathcal{B}^*\hat{V}_k(s)$.

■

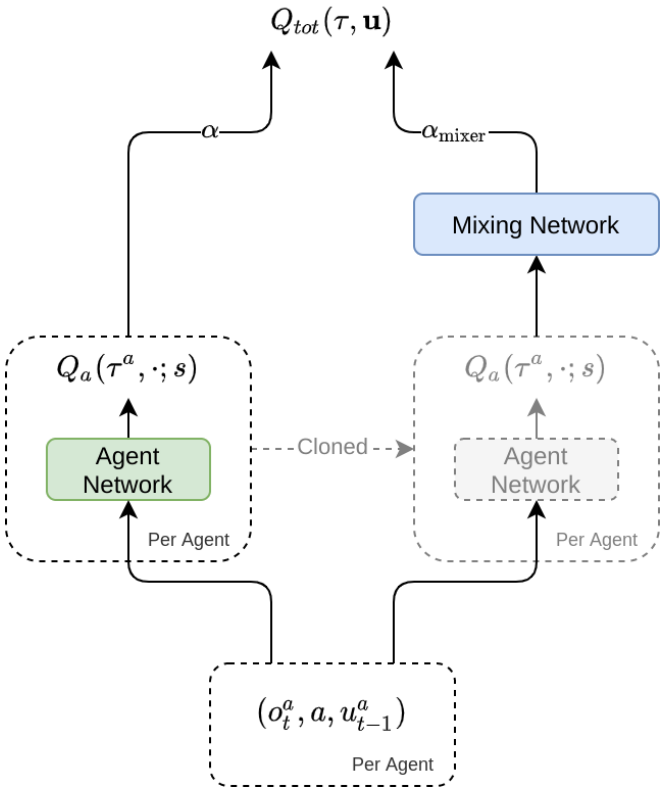


Figure 7.2: GUM Architecture: Ensemble variation with explicit mixing parameters. In the diagram above, a is used to indicate the agent index, whereas α and α_{mixer} indicates the trade-off parameters.

7.4 Experiments

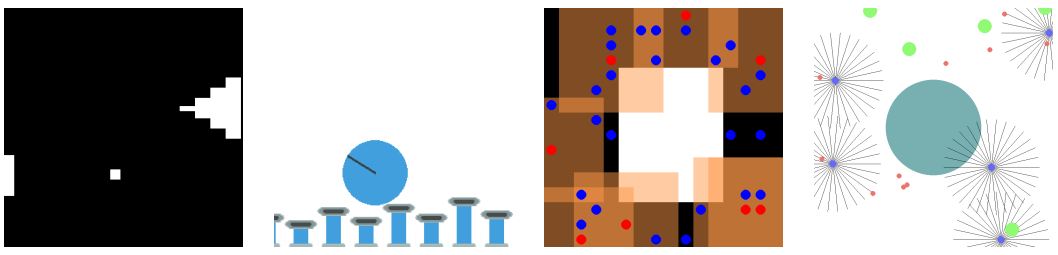


Figure 7.3: Petting Zoo Environments: pong, pistonball, pursuit and water world.

Our MARL experiments are conducted through the benchmark MARL datasets [25, 45, 68] depicted in Figure 7.3.

In this section we empirically justify our approach through various benchmark tasks. To accomplish this, we leveraged various MARL benchmark environments [25, 45, 68], shown in Figure 7.3.

One of the environments is a MARL varient of *Cooperative Pong*, where multiple agents control paddles, with the goal of keeping the puck in play (that is away from the left and right edges). A global positive reward is provided if the puck stays in the environment, and a negative reward otherwise. This environment enforces partial observability by only allowing each agent the ability to observe their half of the environment.

In *Pistonball* it is a cooperative environment where each agent controls a piston with the goal of moving a ball from one side of the screen to the other. Each agent controls a piston which can be moved up and down, where partial observability is enforced through allowing each agent to only observe its immediate area surrounding its piston.

As part of the "SISL" cooperative environments, we explored *Waterworld* and *Pursuit*. In both environments the goal is for the agents to control pursuers which are to capture evaders (in pursuit) or food (in waterworld). In pursuit, the pursuers can only capture the evaders when two or more surround the evaders, whilst in waterworld, the pursuers must concurrently avoid randomly moving poison.

From the "MPE" environments, we examined *Spread* and *Reference* environments, where the goal of both environments is to move agents towards landmarks. In Reference, each agent needs to move to a particular landmark which is not known to the agent itself, but is known by all other agents with the action space including additional actions to directly communicate with other agents, whereas in Spread, the goal is for all agents to cover all agents without collision

7.4.1 Implementation Details

We use the official implementation of MARL algorithms where provided. In particular we leverage the pymarl library for LICA, QMIX, IQL implementations. SEAC and MADDPG was implemented by using the single agent variants in rlkit which was extended to the multi-agent environments. To convert continuous action spaces to discrete, we used gumbel softmax [34, 46], which follows the experiments conducted in SEAC. To construct GUM, we used the official implementation of CQL which uses rlkit, extended to MARL setting.

To ensure consistency across tasks and environments, consistent preprocessing was used as suggested in other MARL benchmark [68]. When evaluating our agents, the tasks were run in separate evaluation environments with 1000 episode rollouts. The hyperparameters used are summarised in Table 7.1.

7.4.2 Hyper-parameters

Hyperparameter	
Replay Buffer Size	1000000
Number of pretraining steps	1000
Steps per Iteration	1000
Discount	0.99
Reward Scale	1
Batch Size	256
Mixer Hidden Sizes	[32, 32, 32]
Policy Hidden Sizes	[64, 64, 64]
Policy Hidden Activation	ReLU
Learning Rate	0.0003
Target Network τ	0.005

Table 7.1: The MARL Experiment hyperparameters

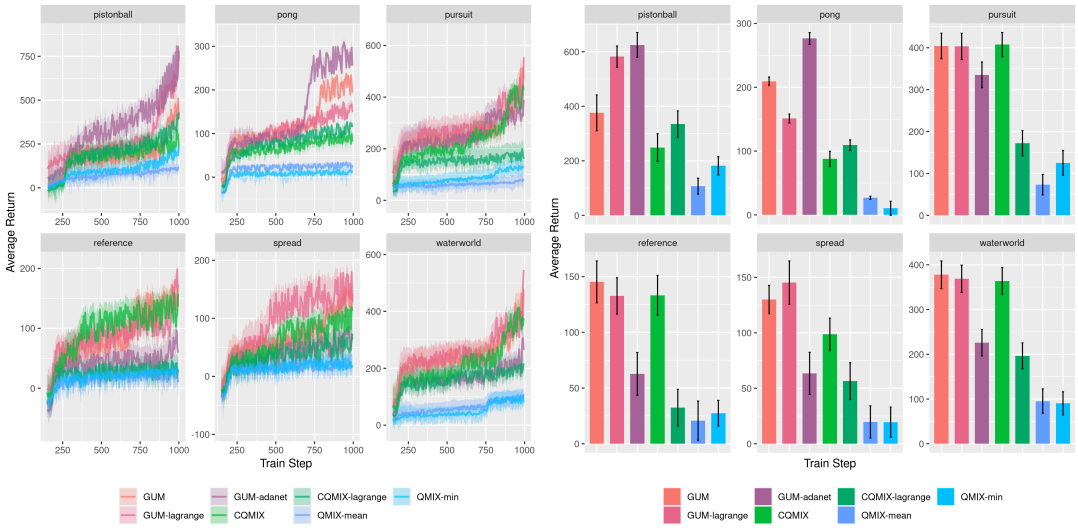


Figure 7.4: GUM Algorithm Ablations. Shown are bar graph with the average return in the last ten iterations with 1000 episode rollouts. The error ranges shown are confidence intervals with one standard deviation. From the results above, we can see the GUM algorithm outperforms other approaches in the ablation studies which demonstrates the theoretical results and motivates our particular choices empirically.

7.4.3 Ablations

For ou ablations, we explore how different architecture choices influence the manner in which an agent generalises. We label the variations as follows: “GUM” algorithm is when the regulariser α representing the trade-off factor is fixed, whilst, if α is a learned parameter we refer this as “GUM-lagrange”. Rather than using implicit methods, if explicit stacking approach is used in a similar manner to AdaNet, we label this approach “GUM-adanet”. As “GUM” is fundamentally CQL with a mixer network, we explore the case where no GUM modifications are made, and instead we blend CQL with QMIX with a fixed trade-off and learned trade-off which has been respectively labelled as “CQMIX” and “CQMIX-lagrange”. To illustrate the effect of naive ensemble approaches, we have explored using the minimum and mean [20, 41, 44] which is respectively labelled as “QMIX-min” and “QMIX-mean”.

Overall, the GUM variations generally perform better than the other algorithms.

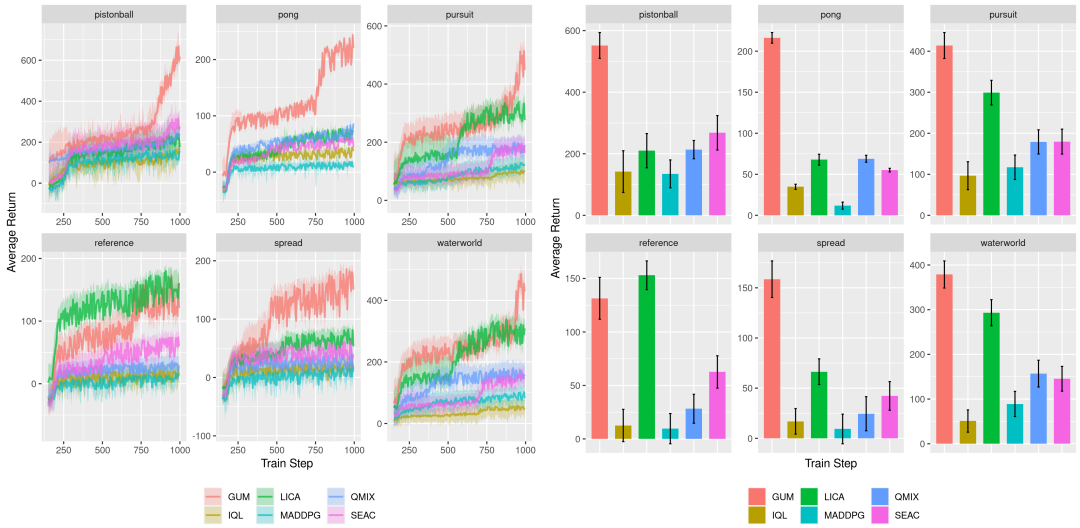


Figure 7.5: Comparison over a variety of benchmark MARL tasks. The average is provided over the last ten steps, with error bars representing one standard deviation.

What is perhaps surprising is the explicit stacking approach performs well in half of the tasks, but is inferior to CQL with QMIX approach for the other environments. Furthermore we can conclude using ensembling techniques in single agent approaches is inferior when used in the multi-agent setting. When comparing the implicit regulariser GUM approaches to the naive CQL with QMIX we can see GUM approach is superior across all environments. This motivates and demonstrates the importance of the regulariser network and unmixing parameter. We hypothesis that the addition of the regulariser alters the update rate between the mixer and independent Q functions in a similar manner to how Polyak averaging is used for the target network.

7.4.4 Experimental Results

When comparing GUM with MARL benchmark and state-of-the-art approaches it performs comparably in cooperative tasks. In this section, all GUM algorithms leverage the variation where the trade-off factors are trainable, and is equivalent to the "GUM-lagrange" variation in the previous section. By examining Figure 7.5, we observe our

approach is comparable or outperforms all other approaches across algorithms regardless whether they use Q-learning or actor-critic approaches.

7.5 Conclusions and Future Work

We introduce GUM which enables ensembling between mixer and independent networks for MARL tasks. We motivate our choices theoretically through examining methods for implicitly and explicitly ensembling the central mixer network when operating in the CTDE framework, and demonstrate the efficacy of our neural network construction empirically through our ablation studies, and superior results compared with state-of-the-art approaches over different benchmark MARL tasks.

Future extensions to this approach could include examining how our approach can be extended to the actor-critic framework and theoretical justifications of our architecture construction approach to enable more effective and efficient MARL algorithms.

8

Conclusion

8.1 Summary

In the first part of this thesis, we focused around the challenge of *detection*. In Chapter 4 and 5 we tackled the challenge around detecting distribution changes in single and multi-agent reinforcement learning. First, in chapter 4 we described a scheme to learn from sub-optimal experiences through leveraging "bad" experiences in the regret framework to constrain policy updates. Experimental results demonstrated that our proposed approach was competitive to existing algorithms and could effectively be used in offline reinforcement learning settings and online even in the face of sub-optimal policies. Chapter 5 explores the multi-agent settings where experiences are shared within the Q-learning setup to enable more efficient training and improve convergence compared with other Q-learning MARL algorithms. Despite the removal of centralised structures, we demonstrate this approach to be superior to the independent Q-learning scenario indicating that through sharing experiences agents have the ability to extrapolate or coordinate despite lack of centralised network structures.

In future we hope to combine our approaches and have the ability to constrain our agents in how policies are executed and trained to constrain policies based on "bad" experiences across multiple agents. As the single-agent scenario is heavily based on actor critic model and the multi-agent scenario revolves around Q-learning, we will need to adapt our algorithms to cater for these differences. This could be through first exploring effective ways to leveraging "bad" experiences in the Q-learning framework and extend to the multi-agent framework, or adapting our approach to the multi-agent setting.

The second part of this thesis revolved around *collaboration*. Chapters 6 and 7 explore the challenge of MARL collaboration through ensembles of agents allow us to unmix agents from their centralised learning structures allowing agents to generate more diverse experiences. Chapter 6 describes the boosting interpretation of residual networks for the neural architecture selection problem, which can be framed and extended to

supervised learning problems such as gradient boosting trees. This idea is leveraged and used in chapter 7 which explores a principled manner to construct MARL algorithms to decouple agents to theoretically enable both centralised and decentralised training concurrently. We demonstrated that this framework provides improvements on certain MARL environments.

As our approach only examines how this is accomplished using QMIX algorithms, additional research could explore how other architectural choices influence performance. Using neural architectural selection could also have interesting implications if we relax the restriction of having a fixed, known number of agents at the beginning of the MARL environment, as it could allow for arbitrary agents which are trained independently and gradually "mixed" together with the other agents. We hypothesis this approach could be used to scale up learning in various MARL tasks.

In the last part of this thesis, we examined *coordination*. Chapter 3 tackles coordination through generating dynamic coordination graph structures which enables flexible structures which can adapt to a wide variety of challenging MARL problems. We demonstrated the flexibility of this representation which allows it to solve complex coordination tasks in the MARL settings, and remains competitive with other state-of-the-art approaches.

One of the exciting prospects of using graph based approaches is the ability to scale out to more agents without altering the underlying graph representation. This would be an exciting avenue to explore, as it would mean a centralised mixing network can be used to train on multiple different MARL tasks, potentially enabling transfer learning and speeding up training.

8.2 Future Work

While all of this is fantastic progress, important open challenges remain:

- One of the key limitations of current approaches presume the number of agents are known and do not consider scenarios where agents can be dynamically added and removed from the environment, and construct networks which can adapt to the constantly changing environment inputs.

- In many applications there may be different levels of coordination which is required as not all agents would act optimally. This is particularly important when agents require to interact with unknown agents such as humans. Being able to generalise their behaviour to cater for this is critical for adapting to applications such as self-driving cars that can properly account for the perspectives of other agents.

- Many cooperative MARL tasks do not leverage a shared global reward. Instead agents are rewarded individually and there may only exist partial common interest. Relaxing this constraint is a under-explored area in the context of cooperative MARL tasks and coordination.

Addressing these limitations in multi-agent learning will profoundly challenge the way we construct autonomous agents and as human beings interact with the world around us.

Acronyms

CTDE Centralised Training with Decentralised Execution. 3, 7, 18, 28, 126, 128

DBR Dual Behavioral Regularisation. 59, 67, 77

DCG Deep Coordination Graphs. 29, 31, 33

Dec-POMDP Decentralised Partially Observable Markov Decision Process. 18

DQN Deep Q-Network. 120, 121

GCN Graph Convolutional Network. 34–36, 39

GUM Greedy UnMixing Network. 8, 126, 139

IQL Independent Q-Learning. 80, 128

LICA Learning Implicit Credit Assignment. 137

MADDPG Multi-Agent Deep Deterministic Policy Gradient. 22, 89, 137

MARL Multi-Agent Reinforcement Learning. 3, 7, 17, 100, 101, 121, 123, 130

MARQ Multi-Agent Regularized Q-learning. 81, 82, 89

MAVEN Multi-Agent Variational Exploration. 31, 42

MLP Multilayer Perceptron. 120, 123

QCGraph Dynamic Q-Value Coordination Graph. 29, 35

QMIX Monotonic Value Function Factorisation. xi, 20, 31, 32, 100, 123, 137, 138

QTRAN Factorize with Transformation. xi, 31, 32

ResNet Residual Networks. 116, 120, 123

RL Reinforcement Learning. 3, 25, 101, 121

SEAC Shared Experience Actor Critic. 7

VDN Value Decomposition Network. 19, 31, 33

Bibliography

[1] A. ABDOLMALEKI, J. T. SPRINGENBERG, Y. TASSA, R. MUNOS, N. HEESS, AND M. RIEDMILLER, *Maximum a posteriori policy optimisation*, in International Conference on Learning Representations, 2018.

[2] M. G. BELLEMARE, W. DABNEY, AND R. MUNOS, *A distributional perspective on reinforcement learning*, in International Conference on Machine Learning, PMLR, 2017, pp. 449–458.

[3] J. BENNETT, S. LANNING, AND N. NETFLIX, *The netflix prize*, in In KDD Cup and Workshop in conjunction with KDD, 2007.

[4] A. BEYGELZIMER, E. HAZAN, S. KALE, AND H. LUO, *Online gradient boosting*, in Advances in Neural Information Processing Systems, 2015, pp. 2458–2466.

[5] W. BOEHMER, V. KURIN, AND S. WHITESON, *Deep coordination graphs*, arXiv preprint arXiv:1910.00091v1, (2019).

[6] J. CASTELLINI, F. A. OLIEHOEK, R. SAVANI, AND S. WHITESON, *The representational capacity of action-value networks for multi-agent reinforcement learning*, in Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1862–1864.

[7] G. CHASLOT, S. BAKKES, I. SZITA, AND P. SPRONCK, *Monte-carlo tree search: A new framework for game ai*, in Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'08, AAAI Press, 2008, p. 216,Äì217.

[8] M. CHEVALIER-BOISVERT, L. WILLEMS, AND S. PAL, *Minimalistic gridworld environment for gymnasium*, 2018.

[9] F. CHRISTIANOS, L. SCHÄFER, AND S. V. ALBRECHT, *Shared experience actor-critic for multi-agent reinforcement learning*, Advances in Neural Information Processing Systems, (2020).

[10] P. CHRISTODOULOU, *Soft actor-critic for discrete action settings*, arXiv preprint arXiv:1910.07207, (2019).

[11] C. CORTES, X. GONZALVO, V. KUZNETSOV, M. MOHRI, AND S. YANG, *AdaNet: Adaptive structural learning of artificial neural networks*, in International Conference on Machine Learning, 2017.

[12] W. DABNEY, M. ROWLAND, M. BELLEMARE, AND R. MUNOS, *Distributional reinforcement learning with quantile regression*, in Proceedings of the AAAI Conference on Artificial Intelligence, 2018.

[13] C. DUGAS, Y. BENGIO, F. BÉLISLE, C. NADEAU, AND R. GARCIA, *Incorporating functional knowledge in neural networks.*, Journal of Machine Learning Research, 10 (2009).

[14] B. EYSENBACH, A. GUPTA, J. IBARZ, AND S. LEVINE, *Diversity is all you need: Learning skills without a reward function*, in International Conference on Learning Representations, 2019.

[15] J. FOERSTER, G. FARQUHAR, T. AFOURAS, N. NARDELLI, AND S. WHITESON, *Counterfactual multi-agent policy gradients*, arXiv preprint arXiv:1705.08926, (2017).

[16] J. FRIEDMAN, T. HASTIE, R. TIBSHIRANI, ET AL., *Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)*, The annals of statistics, 28 (2000), pp. 337–407.

[17] J. FU, A. KUMAR, O. NACHUM, G. TUCKER, AND S. LEVINE, *D4rl: Datasets for deep data-driven reinforcement learning*, 2020.

[18] J. FU, A. KUMAR, M. SOH, AND S. LEVINE, *Diagnosing bottlenecks in deep q-learning algorithms*, in Proceedings of the 36th International Conference on Machine Learning, K. Chaudhuri and R. Salakhutdinov, eds., vol. 97 of Proceedings of Machine Learning Research, PMLR, 09–15 Jun 2019, pp. 2021–2030.

[19] S. FUJIMOTO, E. CONTI, M. GHAVAMZADEH, AND J. PINEAU, *Benchmarking batch deep reinforcement learning algorithms*, arXiv preprint arXiv:1910.01708, (2019).

[20] S. FUJIMOTO, D. MEGER, AND D. PRECUP, *Off-policy deep reinforcement learning without exploration*, in International Conference on Machine Learning, 2019, pp. 2052–2062.

[21] H. GAO AND S. JI, *Graph u-nets*, in International Conference on Machine Learning, 2019, pp. 2083–2092.

[22] S. K. S. GHASEMIPOUR, R. ZEMEL, AND S. GU, *A divergence minimization perspective on imitation learning methods*, in Conference on Robot Learning, PMLR, 2020, pp. 1259–1277.

[23] C. GUESTRIN, M. G. LAGOUDAKIS, AND R. PARR, *Coordinated reinforcement learning*, in Proceedings of the Nineteenth International Conference on Machine Learning, ICML '02, San Francisco, CA, USA, 2002, Morgan Kaufmann Publishers Inc., pp. 227–234.

[24] C. GUESTRIN, S. VENKATARAMAN, AND D. KOLLER, *Context-specific multiagent coordination and planning with factored mdps*, in Eighteenth National Conference on Artificial Intelligence, USA, 2002, American Association for Artificial Intelligence, pp. 253–259.

[25] J. K. GUPTA, M. EGOROV, AND M. KOCHENDERFER, *Cooperative multi-agent control using deep reinforcement learning*, in International Conference on Autonomous Agents and Multiagent Systems, Springer, 2017, pp. 66–83.

[26] T. HAARNOJA, H. TANG, P. ABBEEL, AND S. LEVINE, *Reinforcement learning with deep energy-based policies*, in Proceedings of the 34th International Conference on Machine Learning, D. Precup and Y. W. Teh, eds., vol. 70 of Proceedings of Machine Learning Research, PMLR, 06–11 Aug 2017, pp. 1352–1361.

[27] T. HAARNOJA, A. ZHOU, P. ABBEEL, AND S. LEVINE, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, arXiv preprint arXiv:1801.01290, (2018).

[28] M. HARDT AND T. MA, *Identity matters in deep learning*, in International Conference on Learning Representations, 2017.

[29] H. V. HASSELT, A. GUEZ, AND D. SILVER, *Deep reinforcement learning with double q-learning*, in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, p. 2094,Äì2100.

[30] M. HAUSKNECHT AND P. STONE, *Deep recurrent q-learning for partially observable mdps*, in 2015 aaai fall symposium series, 2015.

[31] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2016), pp. 770–778.

[32] R. HOUTHOOFT, X. CHEN, X. CHEN, Y. DUAN, J. SCHULMAN, F. DE TURCK, AND P. ABBEEL, *Vime: Variational information maximizing exploration*, in Advances in Neural Information Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds., vol. 29, Curran Associates, Inc., 2016.

[33] F. HUANG, J. T. ASH, J. LANGFORD, AND R. E. SCHAPIRE, *Learning deep resnet blocks sequentially using boosting theory*, International Conference of Machine Learning 2018, abs/1706.04964 (2018).

[34] E. JANG, S. GU, AND B. POOLE, *Categorical reparameterization with gumbel-softmax*, in International Conference on Learning Representations, 2018.

[35] P. JIN, K. KEUTZER, AND S. LEVINE, *Regret minimization for partially observable deep reinforcement learning*, in International conference on machine learning, PMLR, 2018, pp. 2342–2351.

[36] Z. JUOZAPAITIS, A. KOUL, A. FERN, M. ERWIG, AND F. DOSHI-VELEZ, *Explainable reinforcement learning via reward decomposition*, IJCAI/ECAI Workshop on Explainable Artificial Intelligence.

[37] G. KE, Q. MENG, T. FINLEY, T. WANG, W. CHEN, W. MA, Q. YE, AND T.-Y. LIU, *Lightgbm: A highly efficient gradient boosting decision tree*, in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio,

H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Curran Associates, Inc., 2017, pp. 3146–3154.

[38] T. N. KIPF AND M. WELLING, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907, (2016).

[39] P. KONTSCHIEDER, M. FITERAU, A. CRIMINISI, AND S. R. BULÒ, *Deep neural decision forests*, in International Joint Conference on Artificial Intelligence, 2016, pp. 4190–4194.

[40] L. KRAEMER AND B. BANERJEE, *Multi-agent reinforcement learning as a rehearsal for decentralised planning*, Neurocomputing, 190 (2016), pp. 82–94.

[41] A. KUMAR, J. FU, M. SOH, G. TUCKER, AND S. LEVINE, *Stabilizing off-policy q-learning via bootstrapping error reduction*, in Advances in Neural Information Processing Systems, 2019, pp. 11784–11794.

[42] A. KUMAR, A. ZHOU, G. TUCKER, AND S. LEVINE, *Conservative q-learning for offline reinforcement learning*, in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds., vol. 33, 2020, pp. 1179–1191.

[43] L. LI, R. MUNOS, AND C. SZEPESVARI, *Toward Minimax Off-policy Value Estimation*, in Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, G. Lebanon and S. V. N. Vishwanathan, eds., vol. 38 of Proceedings of Machine Learning Research, San Diego, California, USA, 09–12 May 2015, PMLR, pp. 608–616.

[44] T. P. LILLICRAP, J. J. HUNT, A. PRITZEL, N. HEESS, T. EREZ, Y. TASSA, D. SILVER, AND D. WIERSTRA, *Continuous control with deep reinforcement learning*, in 4th International Conference on Learning Representations, ICLR 2016, San

Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2016.

[45] R. LOWE, Y. WU, A. TAMAR, J. HARB, O. P. ABBEEL, AND I. MORDATCH, *Multi-agent actor-critic for mixed cooperative-competitive environments*, in Advances in Neural Information Processing Systems, 2017, pp. 6379–6390.

[46] C. J. MADDISON, A. MNIH, AND Y. W. TEH, *The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables*, in International Conference on Learning Representations, 2018.

[47] A. MAHAJAN, T. RASHID, M. SAMVELYAN, AND S. WHITESON, *Maven: Multi-agent variational exploration*, in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., vol. 32, Curran Associates, Inc., 2019, pp. 7613–7624.

[48] V. MNIH, K. KAVUKCUOGLU, D. SILVER, A. A. RUSU, J. VENESS, M. G. BELLEMARE, A. GRAVES, M. RIEDMILLER, A. K. FIDJELAND, G. OSTROVSKI, ET AL., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[49] S. A. MURPHY, M. J. VAN DER LAAN, J. M. ROBINS, AND C. P. P. R. GROUP, *Marginal mean models for dynamic regimes*, Journal of the American Statistical Association, 96 (2001), pp. 1410–1423.

[50] A. NAIR, M. DALAL, A. GUPTA, AND S. LEVINE, *Accelerating online reinforcement learning with offline datasets*, 2020.

[51] G. NEUMANN, *Variational inference for policy search in changing situations*, in Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, Madison, WI, USA, 2011, Omnipress, pp. 817–824.

[52] J. OH, Y. GUO, S. SINGH, AND H. LEE, *Self-imitation learning*, arXiv preprint arXiv:1806.05635, (2018).

[53] F. A. OLIEHOEK AND C. AMATO, *A Concise Introduction to Decentralised POMDPs*, Springer Publishing Company, Incorporated, 1st ed., 2016.

[54] F. A. OLIEHOEK, M. T. SPAAN, AND N. VLASSIS, *Optimal and approximate q-value functions for decentralised pomdps*, Journal of Artificial Intelligence Research, 32 (2008), pp. 289–353.

[55] S. ONTAÑÓN, *The combinatorial multi-armed bandit problem and its application to real-time strategy games*, in Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AAAI Press, 2013, pp. 58–64.

[56] D. PATHAK, P. AGRAWAL, A. A. EFROS, AND T. DARRELL, *Curiosity-driven exploration by self-supervised prediction*, in Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, JMLR.org, 2017, p. 2778,Äì2787.

[57] T. RASHID, M. SAMVELYAN, C. S. DE WITT, G. FARQUHAR, J. FOERSTER, AND S. WHITESON, *Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning*, arXiv preprint arXiv:1803.11485, (2018).

[58] M. D. REID AND R. C. WILLIAMSON, *Composite binary losses*, Journal of Machine Learning Research, 11 (2010), pp. 2387–2422.

[59] S. RUSSELL AND P. NORVIG, *Artificial Intelligence: A Modern Approach*, Series in Artificial Intelligence, Prentice Hall, Upper Saddle River, NJ, third ed., 2010.

[60] J. SCHULMAN, P. ABBEEL, AND X. CHEN, *Equivalence between policy gradients and soft q-learning*, CoRR, abs/1704.06440 (2017).

[61] R. SHAH AND V. KUMAR, *Rrl: Resnet as representation for reinforcement learning*, in International Conference on Machine Learning, PMLR, 2021.

[62] C. SIU, *Residual networks behave like boosting algorithms*, in 2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2019, pp. 31–40.

[63] K. SON, D. KIM, W. J. KANG, D. E. HOSTALLERO, AND Y. YI, *Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning*, arXiv preprint arXiv:1905.05408, (2019).

[64] P. SUNEHAG, G. LEVER, A. GRUSLYS, W. M. CZARNECKI, V. ZAMBALDI, M. JADERBERG, M. LANCTOT, N. SONNERAT, J. Z. LEIBO, K. TUYLS, ET AL., *Value-decomposition networks for cooperative multi-agent learning*, arXiv preprint arXiv:1706.05296, (2017).

[65] R. S. SUTTON AND A. G. BARTO, *Reinforcement Learning: An Introduction*, The MIT Press, second ed., 2018.

[66] R. S. SUTTON, D. A. MCALLESTER, S. P. SINGH, AND Y. MANSOUR, *Policy gradient methods for reinforcement learning with function approximation*, in Advances in neural information processing systems, 2000, pp. 1057–1063.

[67] M. TAN, *Multi-agent reinforcement learning: Independent vs. cooperative agents*, in In Proceedings of the Tenth International Conference on Machine Learning, Morgan Kaufmann, 1993, pp. 330–337.

[68] J. K. TERRY, B. BLACK, M. JAYAKUMAR, A. HARI, L. SANTOS, C. DIEFFENDAHL, N. L. WILLIAMS, Y. LOKESH, R. SULLIVAN, C. HORSCH, AND P. RAVI, *Pettingzoo: Gym for multi-agent reinforcement learning*, arXiv preprint arXiv:2009.14471, (2020).

[69] A. VEIT, M. WILBER, AND S. BELONGIE, *Residual networks behave like ensembles of relatively shallow networks*, in Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, USA, 2016, Curran Associates Inc., pp. 550–558.

[70] N. VIEILLARD, O. PIETQUIN, AND M. GEIST, *Munchausen reinforcement learning*, Advances in Neural Information Processing Systems, 33 (2020).

[71] R. J. WILLIAMS, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine learning, 8 (1992), pp. 229–256.

[72] Y. WU, G. TUCKER, AND O. NACHUM, *Behavior regularized offline reinforcement learning*, arXiv preprint arXiv:1911.11361, (2019).

[73] Z. YING, J. YOU, C. MORRIS, X. REN, W. HAMILTON, AND J. LESKOVEC, *Hierarchical graph representation learning with differentiable pooling*, in Advances in Neural Information Processing Systems, 2018, pp. 4800–4810.

[74] F. YU, H. CHEN, X. WANG, W. XIAN, Y. CHEN, F. LIU, V. MADHAVAN, AND T. DARRELL, *Bdd100k: A diverse driving dataset for heterogeneous multitask learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2636–2645.

[75] E. ZAWADZKI, A. LIPSON, AND K. LEYTON-BROWN, *Empirically evaluating multiagent learning algorithms*, arXiv preprint arXiv:1401.8074, (2014).

[76] M. ZHOU, Z. LIU, P. SUI, Y. LI, AND Y. Y. CHUNG, *Learning implicit credit assignment for multi-agent actor-critic*, Advances in Neural Information Processing Systems, (2020).