A fully-decentralised general-sum approach for multi-agent reinforcement learning using minimal modelling

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Matthew Kruger
1669326

Supervised by Prof. Benjamin Rosman, Dr. Steven James and Mr. Jarrod Shipton

August 31, 2023

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science

Abstract

Multi-agent reinforcement learning is a prominent area of research in machine learning, extending reinforcement learning to scenarios where multiple agents concurrently learn and interact within the same environment. Most existing methods rely on centralisation during training, while others employ agent modelling. In contrast, we propose a novel method that adapts the role of entropy to assist in fully-decentralised training without explicitly modelling other agents using additional information to which most centralised methods assume access. We augment entropy to encourage more deterministic agents, and instead, we let the non-stationarity inherent in MARL serve as a mode for exploration. We empirically evaluate the performance of our method across five distinct environments, each representing unique challenges. Our assessment encompasses both cooperative and competitive cases. Our findings indicate that the approach of penalising entropy, rather than rewarding it, enables agents to perform at least as well as the prevailing standard of entropy maximisation. Moreover, our alternative approach achieves several of the original objectives of entropy regularisation in reinforcement learning, such as increased sample efficiency and potentially better final rewards. Whilst entropy has a significant role, our results in the competitive case indicate that position bias is still a considerable challenge.

Declaration

I, Matthew Kruger, hereby declare the contents of this dissertation to be my own work. This dissertation is submitted for the degree of Master of Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

31/08/2023

Acknowledgements

I want to thank my supervisors, Prof. Rosman, Dr. James and Mr. Shipton, for helping me. Prof. Rosman provided a great deal of encouragement and motivation for me to take reinforcement learning. If not for that, I would not be doing this work. I would also like to thank Dr James and Prof. Rosman for consistently assisting me and transferring their joy and passion for reinforcement learning in their lectures and meetings. If it were not for the assistance of my supervisors and Mr. Beukman this work would be of vastly lesser quality, as they provided constant support and assistance. I want to thank my family, who have always supported me and been willing to listen to all my crazy ideas for all these years, even when they make little sense to them. Lastly, I would like to thank Sandee for always being willing to help me and going out of her way to help me complete forgotten tasks whilst working on this research.

Contents

Chapter 1

Introduction

During their formative years, most individuals engage in various games with friends and family, with one goal being to learn and excel at any new game encountered quickly. For example, when a new edition of a favourite game is released, individuals are likely to want to begin playing it as soon as they have the opportunity. Generally in games, people prefer to spend a short amount of time learning the game initially and start performing well as soon as possible. Additionally, in team-based games, individuals aim to quickly understand the game's mechanics and their teammate's behaviour while collaborating effectively with their teammates. For instance, it is advantageous to have a reliable partner, as random or nonsensical actions can negatively impact the team's performance and morale, leading to frustration among other members. Therefore, players seek dependable partners who consistently make sensible decisions, allowing the team to strategise and concentrate on their contributions to the team, ultimately resulting in a successful and enjoyable gaming experience for all.

Whilst we could only play single-player games, many require more than one player. Similarly, we have reinforcement learning (RL), which focuses on creating agents that can learn to solve or approximately solve specific problems, such as driving a vehicle on an empty highway. However, not all tasks (like games) only have one player; in general, many situations require people to work together. The requirement of multiple agents is true even for RL. For agents to be applicable in the real world, they must acknowledge that many systems require more than one agent [Oroojlooy and Hajinezhad 2022]. Therefore, a subfield of RL that acknowledges the multi-agent nature is multi-agent reinforcement learning (MARL) [Arulkumaran *et al.* 2017a]. Whilst there are many approaches to MARL, we focus on the case where agents must act independently. This independence means agents cannot communicate or explicitly exchange any information during training or execution [Tan 1993; Oroojlooy and Hajinezhad 2022]. We do this because, in real-world tasks, we are often required to act without any communication and must be independent. For example, when we follow friends or family while driving in separate vehicles to a new destination, We cannot talk to them and anticipate their moves; however, we can watch them as they drive and respond effectively, we know that the person we follow will not arbitrarily put their indicators on. Hence, we assume that they are reliable and should they indicate they will turn, we should

do the same, given that our goal is to cooperate and reach the same destination. This dissertation not only delves deep into the nuances of MARL but also pioneers a fresh approach towards addressing its challenges. Through our comprehensive methodology, we highlight the significance of penalising entropy in MARL, shedding light on its impact on agent performance and collaboration in decentralised scenarios.

In this dissertation, we focus on the topic of MARL. Given the challenges and dynamics observed in the gaming analogy mentioned earlier, a pivotal research question arises: "How does penalising entropy in multi-agent reinforcement learning impact the performance and collaboration of agents in decentralized scenarios?" Addressing this question would aid in understanding the implications of penalising entropy and its potential benefits in MARL, especially when compared to traditional entropy maximization methods. Generally, there are two primary approaches to MARL: centralised and decentralised [Papoudakis *et al.* 2019]. Centralised approaches assume that all agents share information (e.g. what they see during training) and collaborate closely [Papoudakis *et al.* 2019]. While this might be advantageous in certain situations, it has significant drawbacks. Specifically, centralised methods may not be scalable, necessitating information sharing that imposes severe constraints, such as requiring all opponents to share their knowledge [He *et al.* 2016; Grover *et al.* 2018; Papoudakis *et al.* 2019]. Additionally, bandwidth constraints pose a serious concern, as real-world agents cannot consistently or continuously transmit and receive information (particularly high dimensional info such as maps or video feeds) due to potential cost implications or environmental factors, such as rural areas with connectivity issues. Given these limitations, we choose to focus solely on the more scalable decentralised approach to MARL [Papoudakis *et al.* 2019]. This allows for greater flexibility and adaptability in a variety of scenarios and helps to mitigate the aforementioned challenges.

Currently, MARL suffers from a similar problem to this "unreliable" teammate from our analogy. In MARL agents may change their policies during training and can be non-stationarity [Hernandez-Leal *et al.* 2017; Papoudakis *et al.* 2019]. This non-stationarity can refer to the scenario where agents' behaviours and the environment may change over time, making it difficult for agents to learn how to act. The standard approach is to encourage agents to explore their environment. This is beneficial in single-agent RL, as it allows agents to learn more rapidly by taking a wide variety of actions in each situation. One way to encourage exploration is through entropy, which is treated as an additional reward for agents during their training [Lowe *et al.* 2017]. According to Ahmed *et al.* [2019], entropy measures the level of uncertainty among agents, which is reflected in their tendency to take different actions. By engaging in a diverse range of actions, agents can explore their environment. However, as indicated in our analogy of unreliable teammates, agents who take various and often random actions may not be the most desirable partners. For the sake of variety, this potential randomness in actions may make it more challenging for other agents to learn how to collaborate. Therefore, we attempt to provide a simple solution: encouraging agents to be more certain about their actions. This certainty is represented by agents increasing the probability of their chosen actions so that they become increasingly likely. Here, we penalise agents for uncertainty about what actions to take in a given state, This penalty encourages agents

to become more certain and less random when choosing their actions, thus allowing them to be more reliable.

We empirically demonstrate that our approach can perform various tasks and improves upon the standard wisdom of encouraging exploration. Additionally, we acknowledge that not all tasks are cooperative and demonstrate that our approach poses no significant detriment in the adversarial case. We examine the adversarial case as opponents may change their behaviours and could act unpredictably or be open to changing their tactics should they be encouraged to explore more. The novelty of our approach lies in the distinct shift from the conventionally accepted wisdom of promoting exploration in MARL. By juxtaposing the traditional methods with our entropy penalisation technique, we underline the potential advantages of steering agents towards more certain actions. The significance of this research stems from its implications on how agents can be made more reliable in multi-agent scenarios, crucial for real-world applications. Our contributions can be summarised as follows:

- We propose an alternative approach of penalising entropy that incorporates an additional measure of entropy.
- We demonstrate that penalising entropy rather than rewarding it can be beneficial.
- We demonstrate that or approach performs (in terms of rewards) as well as a standard base approach in the worst case and can outperform the standard approach in the best case.

We structure the rest of the document as follows. In Chapter 2 we provide background on RL and MARL. Then in Chapter 3 we give an overview of related work in MARL. Following this, we provide and motivate our proposed method in Chapter 4, with Chapter 5 outlining our experimental setup. We present our results in Chapters 6 and 7. Finally, we summarise our work and provide avenues for future work in Chapter 8.3.

Chapter 2

Background

2.1 Outline

The following is provided as the necessary background to understand key aspects of our problem and our proposed approach. Section 2.2 provides the framework for single-agent reinforcement learning and the algorithm we use. We expand upon this framework in Section 2.3 by incorporating multiple agents and providing the cornerstones of MARL. Lastly, in Sections 2.4 and 2.5, we address two essential components of RL non-stationarity and entropy regularisation.

2.2 Single Agent Reinforcement Learning

2.2.1 Markov Decision Process

In single-agent RL, Markov Decision Processes (MDP) provide the mathematical framework for sequential decision-making [Bellman 1957]. A figure that illustrates an MDP is provided in Figure 2.1. To detail the framework, we follow that of Sutton and Barto [2018]. An MDP is the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space $\mathcal{P}$ represents the transition function, $\mathcal{R}$ represents the reward function and γ is the *discount factor*.

The agent and the environment interact with one another at each discrete time step $t \in \mathcal{N}$. In each time step, the agent receives the representation of the environment's current state s_t from the set of all possible states $\mathcal{S}$. At each of these given time steps, the agent takes an action a_t from the set of possible actions $(\mathcal{A})$ available to it in that state. At each time step, the agent receives a reward r_{t+1} according to the reward function $\mathcal{R}(s_t, a_t)$ and transitions to the following state s_{t+1} according to the transition function $\mathcal{P}(s_t, a_t, s_{t+1})$. The reward function specifies the immediate reward that the agent receives for taking action a_t in state s_t, while the transition function specifies the probability of transitioning to state s_{t+1} from state s_t upon taking action a_t. The sequence of states, actions and rewards gives rise to the agent's trajectory, which de-

scribes the agent's passage throughout the episode. An example of how a trajectory would start is as follows:

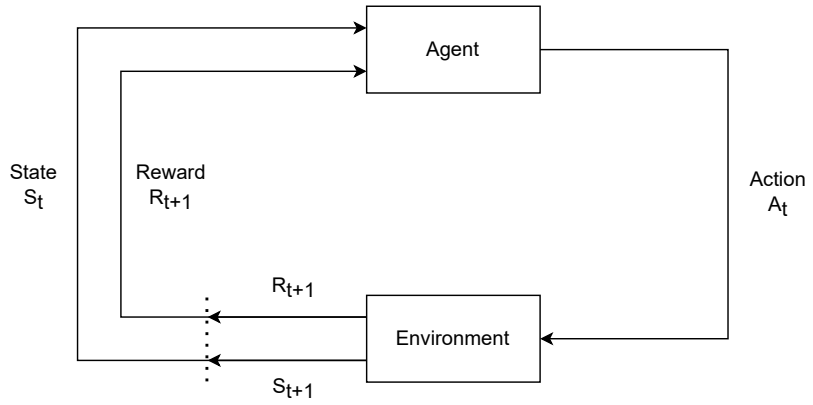


Figure 2.1: Diagram illustrating the interaction between agent and environment in an MDP.

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, \ldots,$$

Where a snapshot of the trajectory at time t would be represented as (s_t, a_t, r_{t+1}). In particular, a finite MDP is characterised by a finite set of states and actions. As a result of the Markov property, the format allows for a discrete probability distribution over the set of potential distributions of these variables, which relies only on the preceding state and action. For example, for any state s' and reward r, we can determine the probability of these values occurring at a specific time step t based on the prior action taken in the previous state. The definition for this conditional probability can be formalised as follows:

$$p(s', r|s_{t-1}, a_{t-1}).$$

The function $p(s', r|s_{t-1}, a_{t-1})$ is the joint probability distribution over the next state s' and the immediate reward r given the previous state s_{t-1} and action a_{t-1} taken by the agent. This function corresponds to the transition and reward functions of the MDP, where the transition function $\mathcal{P}(s, a, s', r)$ determines the probability of transitioning to state s' and receiving reward r when the agent takes action a in state s. The reward function $\mathcal{R}(s, a)$ specifies the immediate reward that the agent receives for taking action a in state s. The function allows us to define the dynamics of a given MDP. Hence, it can be precisely stated that the dynamics are a function of $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$. Additionally, this function p represents a probability function; hence the values are in the range $[0, 1]$. Thus, it is important to note that the following is a result due to its probabilistic nature:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s), \tag{2.1}$$

where $\mathcal{A}(s)$ represents the possible actions in a given state. A necessary concept is that of the expected reward for a state-action pair. The expected reward is a function of the

state and action, which maps to a reward for the agent. Mathematically this is defined as follows:

$$R(s,a) \doteq \mathbb{E}[r_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a). \tag{2.2}$$

An important concept in our MDP, highlighted by Sutton and Barto [2018], asserts that anything not arbitrarily influenced by an agent should be regarded as part of the environment. This concept is essential and is discussed further in Section 2.3.

2.2.2 Agent Rewards and Episodes

An agent's goal in RL is to maximise the cumulative sum of all rewards it receives throughout an episode or over a finite time horizon. This term G_t represents the cumulative sum of rewards from t until a specific final step T. Here T, represents the termination step at which an episode is concluded, and the environment resets to a starting state regardless of how the episode ended. The states that terminate an episode are referred to as terminal states. In reinforcement learning, tasks with terminal states are called episodic tasks. There are cases where no natural end to a task can occur, and the task is perpetual, in which case these are not episodic tasks but instead continuing tasks. For the case of our experiments, we do not consider this scenario. However, it is important to be aware that it is a common problem in RL. We perform *discounting* on the cumulative sum of rewards an agent receives, which is what the agent attempts to maximise. The discounting term is represented by γ, where $\gamma \in [0, 1]$ and is formally referred to as the *discount rate*. Hence, the agent's goal is to learn actions which maximise the expected discounted return as denoted by the following equation:

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.3}$$

The value of γ is vital as it controls the time horizon the agent considers when maximising. A value of zero will result in an agent that is entirely myopic, focusing solely on maximising the immediate reward, whereas a value approaching one will render the agent forward-looking, assigning equal importance to rewards expected in the future.

2.2.3 Policies and Value Functions

In RL, agents act according to a policy which is denoted using π. The policy describes how the agent may act in a given state and, in particular, provides a probabilistic mapping for each action an agent may take in a given state at a specific time step. It is crucial to understand that agents can either have a stochastic policy, which involves a probabilistic mapping of actions, or a deterministic policy, where the agent will take a specific action with a probability of 1 in a given state. This mapping is represented as $\pi(a|s)$, where $A_t = a$ represents the action taken at time step t and $S_t = s$ which is the state at that time step. Hence, this means that the policy forms a distribution over potential actions for the current state at a particular time step. A second component is necessary to help describe the perceived value of a state. In this case, we use a state-value function for policy π, denoted by $v_\pi(s)$, which specifies the expected return of a

given state with the agent following policy π. This function is formally defined in the case of MDPs as follows:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|S_t = s\right], \forall s \in \mathcal{S}, \tag{2.4}$$

where the value function represents the expected value of a given state with the agent following a specific policy π at any given time step t. Given that we may want to see the value of a given state should the agent take a specific action a, we have another function: the action-value function for policy π. This function represents the expected return for an agent, which starts in state s, takes a particular action a in that state and then proceeds to follow policy π afterwards. The function is mathematically denoted as the following:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]. \tag{2.5}$$

Consequently, stemming from these ideas is the crucial concept involving *optimal* policies, state-value functions, and action-value functions. Value functions allow us to partially order policies and hence allow us to "rank" policies. For example, for two distinct policies π and $\pi^{'}$ we can say π is better than or equal to $\pi^{'}$ *iff* the expected return of π is greater than or equal to $\pi^{'}$ for all states. Fundamentally, this is represented as follows:

$$\pi \geq \pi^{'} \iff v_\pi \geq v_{\pi'}, \forall s \in \mathcal{S}. \tag{2.6}$$

There will always be an optimal policy; however, it may not be unique. This policy is the policy which is as good as or better than all other policies. Despite this, all optimal policies are denoted as π_*, and they all have the same state-value function, which is called the *optimal state-value function* and is denoted as v_*, which is defined as follows:

$$v_*(s) \doteq \max_\pi v_\pi(s), \forall s \in \mathcal{S}. \tag{2.7}$$

Furthermore, optimal policies share another component, which is that of the *optimal action-value function*, which is denoted as q_* and is mathematically defined as:

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a). \tag{2.8}$$

It is not always possible for an agent to learn the optimal policy. This inability may be due to various factors, such as the agent's memory, the state-space's complexity or the computational resources available to an agent. Hence, we need to introduce the notion of an *approximate optimal policy*.

2.2.4 Policy Gradient Methods

One robust set of methods in RL is Policy Gradient Methods, which aim to directly optimise the policy function in order to maximise the expected reward. These methods

use parameterised policies, which means they do not need to use a value function; however, they can use one to learn policy parameters. The parameters of an agent's policy are represented using $\boldsymbol{\theta} \in \mathbb{R}^d$, where represents the number of parameters in the policy. Hence, the policy of an agent can be described as the following:

$$\pi(a|s, \boldsymbol{\theta}) = p(A_t = a|S_t, \boldsymbol{\theta}_t = \boldsymbol{\theta}), \tag{2.9}$$

which represents the probability that an agent at time step t in state s will take action a given its parameters $\boldsymbol{\theta}$. When approximating the value function, if the method learns said value function as well, we denote the value function's weight vector as $\boldsymbol{w} \in \mathbb{R}^d$. The value function becomes a function of the state and $\boldsymbol{w}$ and is denoted as $\hat{v}(s, \boldsymbol{w})$. In general, an agent learns according to a given cost function that measures its performance which it seeks to maximise. This cost function is represented using $\boldsymbol{J}(\boldsymbol{\theta})$, with the algorithm performing gradient ascent to maximise the function. The equation governing the update to this function is as follows:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)}, \tag{2.10}$$

where α is the learning rate, and $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ is a stochastic estimate approximating the cost function's performance in accordance with θ at a given time step t. One of the first policy gradient methods is REINFORCE which utilises the empirical return of a given trajectory in its update and, as a result, suffers from high variance in its gradients due to using the empirical mean from each trajectory during its updates [Williams 1992; Arulkumaran *et al.* 2017b]. A standard solution to this is to subtract a baseline, such as the mean return over multiple episodes [Konda and Tsitsiklis 1999; Schulman *et al.* 2015b; Arulkumaran *et al.* 2017b]. However, alternatives to this baseline exist in value-function methods; instead they learn a value-function and use it as a baseline [Konda and Tsitsiklis 1999; Schulman *et al.* 2015b; Arulkumaran *et al.* 2017b].

Methods which learn to approximate both the value function and the policy function are referred to as *actor-critic* methods. Here an actor is the agent's policy, with the critic being the value function. In general, neural network architectures are used when performing these methods [Mnih *et al.* 2016; Schulman *et al.* 2015a 2017b]. The use of these neural networks in RL is referred to as *Deep Reinforcement Learning*. The success of Deep Reinforcement Learning is attributed mainly to the powerful function approximation of neural networks, its ability to effectively deal with the *curse of dimensionality* and how it has introduced new approaches to RL [Arulkumaran *et al.* 2017b]. A common method of parameterisation for agents are neural networks, where the networks parameters are represented using $\boldsymbol{\theta}$, which represents the current weights of the network.

2.2.5 Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is a widely known algorithm used in standard RL and MARL settings [Schulman *et al.* 2017b; Terry *et al.* 2022 2020b; Zhou *et al.* 2020]. As an actor-critic method, it offers multiple algorithmic design choices [Hsu *et al.* 2020].

PPO enhances the efficiency of the Asynchronous Actor-Critic (A3C) algorithm, a successful RL method, by incorporating elements of asynchronous actor-critic methods such as those in Mnih *et al.* [2016] that allow parallel execution.

PPO was developed as a method to simplify and improve upon Trust Region Policy Optimisation (TRPO)[Schulman *et al.* 2015a 2017b]. An important aspect of PPO is the use of the Kullback-Leibler (KL) divergence [Kullback and Leibler 1951], which measures the difference between the current policy during its update procedure and the original policy before the update. The KL divergence enables PPO to reuse a set of sample trajectories for multiple policy updates while preventing policy weight updates from exceeding a given threshold. PPO also incorporates an advantage estimate $\hat{A}_t$, introduced by Mnih *et al.* [2016], that allows a policy to run for T time steps and perform iterative updates using collected samples. The advantage estimate approximates the current estimate of the state's value following the agent's current policy and helps to reduce the variance in the update without requiring information after the time step T. This advantage estimate uses a parameter λ, which is a part of the Generalised Advantage Estimation (GAE) technique, that controls the trade-off between bias and variance in the estimation of the advantage function. Specifically, a λ close to 0 results in a high bias but low variance estimate (more reliant on immediate differences in value estimates), while a λ close to 1 results in a low bias but high variance estimate (incorporates more future differences in value estimates). The equation is as follows:

$$\begin{aligned} &\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + \cdots + \gamma^{T-t+1}\lambda^{T-t+1}\delta_{T-1}, \\ &\text{where } \delta_t = r_t + \gamma V_{s_{t+1}} - V s_t. \end{aligned} \tag{2.11}$$

Here λ represents the discounting.

$$D_{KL}(P||Q) = \sum_{x \in \mathbf{X}} P(x) log(\frac{P(x)}{Q(x)}) \tag{2.12}$$

Equation 2.12 provides the KL divergence of two discrete probability distributions P and Q which are defined over the same sample space ($\mathbf{X}$), where $D_{KL}(P||Q)$ defines the relative entropy between these two distributions. An important component PPO inherited from TRPO is that of its surrogate objective function given by the equation below.

$$L^{surrogate} = \max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t - \beta D_{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]. \tag{2.13}$$

Here, $\beta \in \mathbb{R}$ represents a coefficient that controls the important of the KL Divergence in the unconstrained optimisation problem. In the following equation, we represent $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ as $r_t(\theta)$ to help simplify the following equations. Lastly, D_{KL} represents the KL divergence between the current and old policies. This equation is modified further when constructing the final PPO objective function, which is as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{2.14}$$

where the value of ϵ is a hyperparameter that helps prevent excessively large updates, $\hat{A}_t$ represents the estimate of the advantage function at time step t. The first term in

Algorithm 1 PPO

1: **for** iteration=1,2,... **do**
2: **for** actor=1,2, ...N **do**
3: Run policy π_θ in environment for T time steps
4: Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
5: **end for**
6: Optimise surrogate L in Equation 2.15 w.r.t θ, with K epochs and mini-batch size $M \leq NT$
7: **end for**

the PPO loss function is the unclipped objective, which can allow for a larger update to the policy, but it is constrained by ϵ and clipping, which prevent it from deviating too far from the previous policy. A fundamental way to describe Equation 2.14 is that the second term in the minimum is that of the clipped objective function, which is considered more "conservative" as it attempts to prevent excessively large policy updates, updates which may be detrimental to the performance of the agent. Clipping refers to bounding the value in a range, as defined in this equation by $1 - \epsilon$ and $1 + \epsilon$. Hence, the combination of these two provides a lower bound on the objective function, and it allows for larger changes in the policy when the objective function improves and allows it to be conservative about the level of policy change when it is detrimental to the objective function. Here the clipped surrogate loss function in Equation 2.14 is incorporated into the following equation which PPO utilises and attempts to maximise during each iteration.

$$L_t(\theta) = \hat{\mathbb{E}}_t[L^{CLIP}_{t(\theta)} - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t)]. \tag{2.15}$$

Here, S represents an entropy bonus term, which is a component of entropy regularisation. This term encourages exploration while penalising policies that are overly deterministic. The squared loss $(V_\theta(s_t) - V_t^{targ})^2$ signifies the discrepancy between the target function (the agent's current estimate of the state's value) and the critic's value function. Lastly, c_1 and c_2 denote coefficient values, with c_1 weighting the importance of the difference in the value function and the estimated value function, and c_2 determining the importance of entropy in the objective function. PPO integrates an essential concept from Mnih *et al.* [2016] that allows the algorithm to run for T time steps, where T is much less than an episode's length and uses the collected set of samples to perform a batch update. This update is typically done using mini-batches from the batch. The PPO algorithm in an Actor-Critic format, as given by Schulman *et al.* [2017b] is provided in Algorithm 1. Here, each iteration (Line 1) consists of N parallel actors (Line 2), which act and generate a policy rollout of T steps of data (Line 3). The cumulative set of NT data points allows us to construct our surrogate loss function and optimise it (Line 5) using stochastic gradient descent for K epochs.

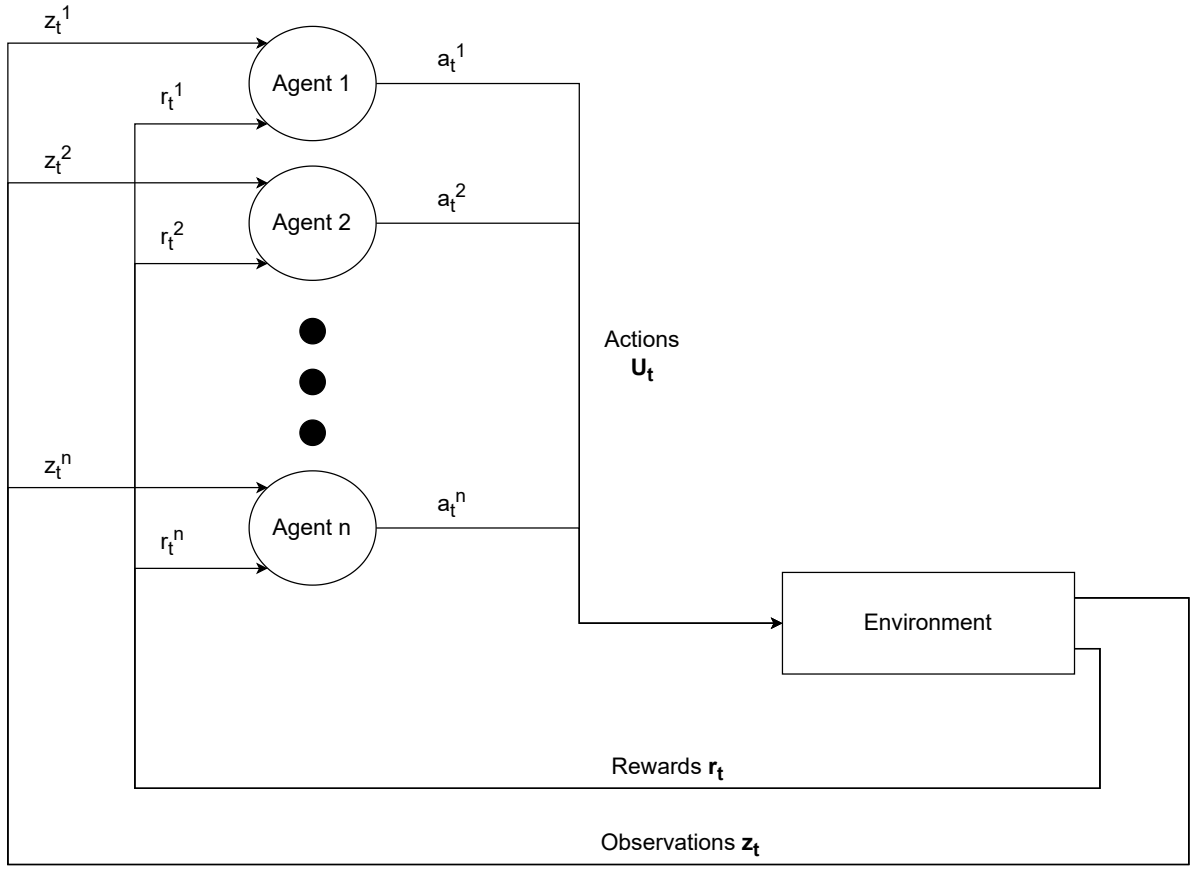


Figure 2.2: The provided figure delineates a Markov Decision Process (MDP) within a multi-agent environment, with n agents performing actions concurrently. In this context, t denotes the present time step, z_t^i signifies the observation of the ith agent, r_t^i corresponds to the ith agent's reward, and a_t^i symbolises the action taken by the ith agent. Moreover, the terms $\mathbf{z}_t$, $\mathbf{r}_t$, and $\mathbf{U}_t$ refer to a vector of observations, a vector of rewards, and the joint-action vector of all agents, respectively. During each time step, all agents take actions in unison, resulting in the formation of a joint-action vector. Subsequently, at the next step, each agent acquires an observation and a reward. These are derived from the vectors of observations and rewards, which contain the specific observation and reward for each individual agent.

2.3 Multi Agent Reinforcement Learning

2.3.1 Partially Observable Stochastic Games

In this work, we consider the *partially observable stochastic game* (POSG) formalism of Multi-agent RL (MARL), which is defined by a tuple $(I, S, O, A, r, \mathcal{T})$ [Peshkin *et al.* 2000; Hansen *et al.* 2004]. Here, I represents the set of n agents, S is the set of states, $A = A_1 \times A_2 \times \cdots \times A_n$ represents the set of all actions, $r = (r_1, r_2, \ldots, r_n)$ is the vector of each of the agents' rewards where $r_i : S \times A \times S \to \mathcal{R}$ and represents the reward function for the i^{th} agent. Since we are in a partially observable setting, the agents do not have access to the full state $s \in S$, instead receiving an observation

$z_j \in O_j$; with $O = O_1 \times O_2 \times \cdots \times O_n$. Lastly, $\mathcal{T} : S \times A \times S \times O \rightarrow [0, 1]$ represents the probability distribution over the following states and joint observations given the current state and actions taken by all the agents. A simple illustration of the problem is given in Figure 2.2. Each agent is represented by a policy $\pi_j(a|z_j)$ conditioned on their observations, which determines the action the agent will take in a given scenario. We use u_t to denote the set of actions taken by the agents at time step t. We note that in MARL, the joint policy of the agents specifies how each agent selects its action based on the state of the environment and the actions of the other agents, creating a joint action-value function represented by $Q_\pi(z_t, u_t) = E_\pi(R_t[z_t, u_t])$, where R_t denotes the discounted cumulative reward over a finite time horizon T and u_t is a vector that contains the actions of all agents taken at time step t. The agents' goals are to learn a joint optimal policy π_*, which is better than or equal to any other policy as denoted by the following:

$$Q_{\pi_*}(z_t, u_t) \geq Q_\pi(z_t, u_t), \forall \pi.$$

Finally, we note each agent's value and policy function is parameterised by θ and ϕ, respectively, and that θ^j and ϕ^j represents the j^{th} agent's policy and value function parameters, respectively.

2.3.2 Information Structure

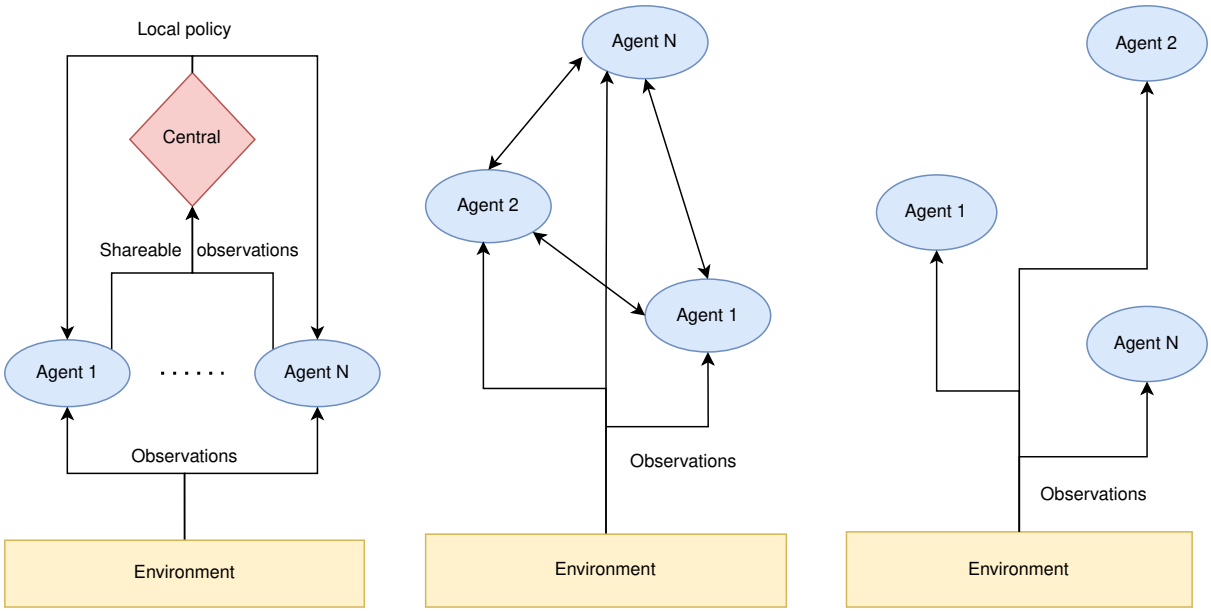


Figure 2.3: This figure shows the three information structures used in Multi-Agent Reinforcement Learning (MARL): fully-centralised, CTDE, and fully-decentralised (from left to right). These structures illustrate how decisions and information are shared between agents. In the fully-centralised structure (left), a single actor controls all agents and makes decisions based on all observations. The CTDE structure (middle) includes multiple central controllers, one for each agent, and allows for information sharing during training, but agents act based on local observations. The fully-decentralised (right) structure involves agents receiving only their own local observations.

In MARL, agents typically receive observations rather than the global state. These observations represent what the agents can observe in the environment and are MARL's equivalent to the state-space for agents in single-agent RL. The architecture of the underlying approach to sharing information and networking agents in MARL can differ. There are three primary ways to structure the flow of this information, and they are as follows [Nguyen *et al.* 2020; Zhang *et al.* 2021]:

1. Fully-centralised
2. Centralised-training-decentralised-execution (CTDE)
3. Fully-decentralised

Each of these approaches has specific advantages and disadvantages. Fully-centralised approaches use a single network which aggregates all information from each agent and shares it amongst the agents [Zhang *et al.* 2021]. This approach helps to alleviate non-stationarity as the centralisation allows for the sharing of all relevant information; however, it comes at the cost of a more computationally complex problem [Oroojlooy and Hajinezhad 2022]. In particular, this complexity can be restraining as it relies on a single controller controlling all agents and hence the number of actions can exponentially increase as more agents are added and agents need to aggregate all local observations with this single controller [Oroojlooy and Hajinezhad 2022]. The exponential increase in complexity as the number of agents increases is referred to as the *combinatorial nature* of MARL [Hernandez-Leal *et al.* 2019; Zhang *et al.* 2021].

An alternative to this approach are fully-decentralised methods. These methods do not assume access to other agents' information or observations [Hernandez-Leal *et al.* 2019]. Moreover, these methods are more scalable than fully-centralised methods, as they can be implemented in a distributed fashion, with each agent making decisions independently based on local observations. However, they are also more susceptible to non-stationarity, as agents may have incomplete information and lack explicit coordination [Papoudakis *et al.* 2019; Hernandez-Leal *et al.* 2019; Oroojlooy and Hajinezhad 2022]. One subcategory of these methods is "independent learners", which are algorithms that do not necessarily acknowledge that other agents are involved [Hernandez-Leal *et al.* 2018; Zhang *et al.* 2021; Oroojlooy and Hajinezhad 2022]. In particular, independent learners treat MARL problems as single agent problems.

Finally, some methods are centralised during training and then decentralised during execution and these methods are referred to as CTDE (centralised-training-decentralised-execution) [Hernandez-Leal *et al.* 2018 2019; Nguyen *et al.* 2020; Rashid *et al.* 2020; Oroojlooy and Hajinezhad 2022]. These methods have become prevalent in modern MARL as they strike a compromise between scalability and dealing with inherent non-stationarity. However, these methods rely on a centralised critic during runtime or parameter sharing [Terry *et al.* 2022 2020b]. Despite this, there is a preference for fully-decentralised methods by researchers such as Zhang *et al.* [2021], as simulators for environments may not always be available. Therefore, we may not always be able to make use a centralised critic which is able to share all information amongst all agents during their training. This drawback primarily arises in new or unexplored domains, in situations where a simulator is not available for use, or when creating an accurate

simulator is prohibitively complex, such as perfectly replicating the behaviour of every ant in a large ant colony. The choice of information structure has integral role in the type of game the agents are involved in.

2.3.3 Types of Games

MARL agents can be involved in three primary types of games, which create a strong link between MARL and its interaction with Game Theory [Shoham *et al.* 2007]. They are as follows:

1. Cooperative
2. Competitive
3. General-sum

Firstly, cooperative games require agents to work together and share a common reward function [Zhang *et al.* 2021]. An example of a cooperative game is the StarCraft Multi-Agent Challenge (which makes use of the StarCraft II Learning Environment), where agents on a team must work together to defeat an AI foe [Vinyals *et al.* 2017]. Typically, in cooperative games, rewards are shared between each agent. However, credit assignment is a sub-problem in this field which aims to assign each agent a specific reward based on their impact on the total final reward [Hernandez-Leal *et al.* 2018 2019]. Secondly, we have competitive games, also known as competitive games [Zhang *et al.* 2021]. In these games, agents view one another as opponents and compete against one another. Generally, these games are "zero-sum", as one agent's reward is equivalent to another agent's loss. Hence, when the rewards for each agent are summed together, the environment has a reward of zero, thus, zero-sum [Foerster 2018]. One example of this type of game is chess, where if one player wins, the other loses, with a draw being a "zero" reward outcome. Lastly, there are environments that mix cooperative and competitive cases. These environments are called "general-sum"; unlike zero-sum, the reward in the system can be non-zero [Foerster 2018; Zhang *et al.* 2021]. These types of games are also known as mixed-setting games [Hernandez-Leal *et al.* 2019; Zhang *et al.* 2021]. Poker is a classic example of a general-sum game, where each player's objective is to win the pot of money, and each player's gain or loss is dependent on the decisions and strategies of all players, rather than being equal or opposite.

2.4 Non-stationarity

Non-stationarity is a recognised challenge in RL, but it becomes even more pronounced in MARL. This challenge emerges when the environment or the problem setting contains dynamically evolving elements or behaviours, such as various agents learning and adaptively changing their policies. In MARL, non-stationarity arises from the "Moving Agent" problem. The Moving Agent problem refers to when more than a single agent is learning. The agents may develop policies that are based on another agent. A concise way to phrase this problem is given by Papoudakis *et al.* [2019], where this is referred to as a "ringing agent" problem. In this description, the "ringing agent" problem refers to

the case when an agent is assumed to be static (i.e., not learning), and other agents develop policies based on this assumption. However, in reality, this agent is also learning, resulting in a continually changing policy. This problem is important to note because it contributes to the non-stationarity problem in MARL. As a result, by the time one agent has created a policy focused on another, that agent's policy may have drastically changed, causing the agent to have a suboptimal policy. This process continues as the agents interact, hence it suffers from the ringing agent problem. In MARL, agents must account for the behaviours of other agents, which leads to the violation of the stationary Markovian property. This property states that the reward and current state should depend solely on the previous state and action taken by the agent. However, in the context of MARL, this assumption no longer holds true due to the interactions amongst agents [Zhang *et al.* 2021].

Centralised methods are one way to account for non-stationarity, as stated by Hernandez-Leal *et al.* [2018 2019]; Papoudakis *et al.* [2019]. This success is potentially because all agents' information is shared and thus taken into account by a central controller. In particular, the fully-decentralised case suffers to a greater extent from non-stationarity, given the information structure for those agents [Papoudakis *et al.* 2019; Zhang *et al.* 2021]. As a result, fully-decentralised agents are more likely to overlook crucial information related to the participating agents.

2.5 Entropy Regularisation

Before delving into the role of entropy regularisation, first consider a type of exploration strategy known as "$\epsilon - greedy$" [Sutton and Barto 2018]. This strategy primarily involves taking "greedy" actions that exploit the agent's current understanding of the environment—specifically, which action is believed to yield the best Q-value at that moment. However, until the agent possesses a comprehensive and accurate representation of the Q-value for each state, it might not always discern the truly optimal action in every scenario. To counterbalance this, we introduce "exploratory" actions, which are taken with a probability $\epsilon \in [0, 1]$. These actions aim to refine the agent's estimate of the value of non-greedy actions, typically resulting in a higher cumulative reward over the course of an episode, especially as training progresses. The concept of entropy regularisation was introduced to enhance exploration strategies in modern reinforcement learning. The role of entropy regularisation is to help agents explore [Mnih *et al.* 2016; Schulman *et al.* 2017ba], which is critical to discover good policies during optimisation [Ahmed *et al.* 2019]. Entropy regularisation does this by encouraging agents to take a more diverse set of actions by preventing them from becoming deterministic too rapidly [Williams and Peng 1991; Mnih *et al.* 2016; Ahmed *et al.* 2019]. This diversity results from an additional term in the agent's objective function [Mnih *et al.* 2016; Schulman *et al.* 2017b]. The additional term serves as a reward for the agent when it takes an action with a lower probability. Hence, the entropy term helps in encouraging the agent to explore. Entropy regularisation was first used by Williams and Peng [1991] and has been a common component of various other works [Mnih *et al.* 2016; Schulman *et al.* 2017b; Lowe *et al.* 2017; Liu *et al.* 2019; Jaques *et al.* 2019; Zhou *et al.* 2020]. The most prevalent measure of entropy in RL is that of Shannon Entropy

which was proposed by Shannon [1948] as a measure of information gain in Information Theory. The following is the equation for Shannon entropy in the single agent case using the MDP framework:

$$H(\pi(a_t|s_t)) = -\sum_i p(a_t^i|s_t)\log(p(a_t^i|s_t)). \tag{2.16}$$

In Equation 2.16, s_t denotes the state at time t, a_t denotes the action at time step t, and the summation is taken over each potential action that the agent could take at this time step as indicated by the indexing variable i, where $p(a_t^i)$ represents the probability of agent i taking action a at time step t. As can be seen in Equation 2.16 the value is maximised when all actions are equally likely and minimised should the agent become deterministic. One commonly stated reason for this technique, as given by Mnih *et al.* [2016] is that "adding the entropy of the policy π to the objective function improved exploration by discouraging premature convergence to suboptimal deterministic policies." An example of an equation that incorporates entropy regularisation in Policy Gradient methods is given below:

$$L = \nabla_{\theta'}\log\pi(a_t|s_t,\theta)(R_t - V(s_t|\theta_v)) + \eta\nabla_{\theta'}H(\pi(s_t|\theta)). \tag{2.17}$$

In the context of deep reinforcement learning using neural networks for function approximation, the objective is often to minimise a loss function by iteratively updating the network parameters. The gradients, represented by the nabla symbol, indicate the direction and magnitude of changes required for these parameters to reduce the loss. Thus, the loss in Equation 2.17 represents the gradient with respect to the policy and value function parameters, guiding the learning process. The equation is composed of three terms. The first term, $\nabla_{\theta'}\log\pi(a_t|s_t,\theta)$, represents the gradient of the log probability that the agent takes action a_t in state s_t while using the policy parameterised by θ. The second term, $(R_t - V(s_t|\theta_v))$, represents the advantage function, which is the difference between the actual reward received (R_t) and the estimated reward ($V(s_t|\theta_v)$) for that state. The third term, $\eta\nabla_{\theta'}H(\pi(s_t|\theta))$, is the entropy regularisation term, where $H(\pi(s_t|\theta))$ represents the entropy based on the agent's policy. The coefficient η controls the strength of the regularisation, and θ_v represents the parameters of the value function. In Section 3.6, we go into greater detail about Entropy Regularisation.

Chapter 3

Related Work

3.1 Outline

We structure the related work as follows. Initially we examine our particular domain problem and provide a brief overview as to why an alternative approach is necessary in Section 3.2. We then examine three particular settings in MARL and the algorithms which are designed for each of these settings. These settings are the cooperative case in Section 3.3, the general-sum and competitive cases in Section 3.4. Following this, we provide a more detailed overview of PPO in Section 3.5, breaking down some augmentations others have made to PPO. In particular, we examine some modifications that allow PPO to function as a standard MARL algorithm. Thereafter, in Section 3.6 we examine entropy regularisation and its current role in RL.

3.2 Domain Problem

In MARL, various algorithms are designed with a specific setting in mind. For example, algorithms that seek to incentivise another agent's exploration to encourage cooperation [Jaques *et al.* 2019] will not necessarily work in competitive settings. Alternatively, algorithms which are selfish are only concerned with themselves and their performance [Foerster *et al.* 2017b] are unlikely to be as successful in tasks where the team reward is the focus. Other approaches focus on hand-crafted methods to learn in MARL, such as those of Albrecht and Ramamoorthy [2015]. However, many algorithms utilise deep reinforcement learning to learn in MARL [Chu and Ye 2017; Gupta *et al.* 2017; Lowe *et al.* 2017; Foerster *et al.* 2017b 2018]. We break down some of these algorithms and identify a clear gap in current approaches. Currently, the primary focus in MARL is in centralised-training-decentralised-execution methods. These methods help overcome the non-stationarity that exists in MARL. Furthermore, these methods allow for additional advantageous assumptions, which are typically in the form of other agents' exact gradients, responses or observations. This extra data allows for other algorithms to incorporate a rich set of data into their method. However, methods which do this are limited in their applicability as they will require simulators which are not always available. They also require information to be transferred, which due to bandwidth

constraints, limit these approaches in the real world. It is important to note that this information could often be high dimensional for example, image data from an agent's observation which may have a high degree of granularity (for example multiple frames from a video feed). In cases where we are referencing the transfer of this information in a real world setting we reference it as *communication*. Lastly, they may assume that an opposing agent, in a competitive game would allow access to such data. Hence, this assumption is not realistic as no opponent would share vital information to their opponent. Whilst methods that are fully-decentralised do exist they typically use hand-crafted methods or assumptions which can be prohibitive such as that of Albrecht and Ramamoorthy [2015] and Jaques *et al.* [2019]. We discuss the specific limitations and assumptions for these approaches in Sections 3.3.2 and 3.4.3. One of the strongest aspects of these methods is their scalability.

Therefore, in our domain, we desire agents that make minimal assumptions about all other agents. These agents should perform in a fully-decentralised fashion to avoid bandwidth issues and the need for simulators. Lastly, they must not use hand-crafted techniques to make the algorithms widely and readily applicable to a variety of settings.

3.3 Cooperative Methods

The cooperative setting is an interesting and vital component of MARL. For MARL to be widely and readily applicable, there are numerous circumstances where agents must be able to work together [Foerster *et al.* 2018]. This section examines influential MARL algorithms that focus on the cooperative setting.

3.3.1 Fully-Centralised and CTDE Methods

Agent Specific Credit Assignment

A common issue in MARL is assigning credit to each agent. Agents act individually and receive a reward; sometimes, however, they receive a team reward, which all agents receive. This team reward can make it challenging for agents to know how responsible they are for the reward they receive. However, there are ways to overcome this challenge; one potential method is Counterfactual Reasoning. Counterfactual thinking has its roots in psychology, where it refers to the tendency for humans to think of and create possible alternatives to events that have already occurred, and focus on "what might have been" [Roese 1997]. This allows examination into how a particular change in choice may have resulted in different consequences. One algorithm, *Counterfactual Multi-Agent Policy Gradients* (COMA), by Foerster *et al.* [2018] serves as an illustration of the key attributes to consider in this problem. COMA uses *difference rewards* to provide credit to each agent [Foerster *et al.* 2018; Nguyen *et al.* 2020]. Prior work by Wolpert and Tumer [2001] inspired the idea of these difference rewards. Their purpose is to compare the reward the agent receives for taking a specific action rather than having taken a default action. This difference reward assigns credit to each agent for their influence on the final reward; hence giving agents credit for their impact. Difference rewards are a valuable method because agents must have the ability to work together

and be able to successfully perform their own roles to succeed in this setting. One of the issues that COMA overcomes is estimating what the difference reward should be, which is done via a centralised critic [Foerster *et al.* 2018]. COMA does this because the centralised critic is trained using all agents' observations and the joint action of all agents in the game; denoted as $\boldsymbol{u}^a$. This joint action represents each agents' actions at a given time step. Therefore, COMA compares the Q-value function for agents taking a specific set of actions $\boldsymbol{u}^a$. Alternatively, $\boldsymbol{u}^{-a}$ denotes the set of actions taken by all other agents excluding agent a. This creates a different baseline for each agent which is used when performing updates, the baseline that is used for a given agent i is as follows:

$$\hat{A}^i(s, \boldsymbol{u}) = Q(s, \boldsymbol{u}) - \sum_{u^i} \pi^a(u^i|\tau^i) Q(s, (\boldsymbol{u}^{-i}, u^i)). \tag{3.1}$$

Here, $\hat{A}^i(s, u^i)$ is a separate baseline that each agent will compute, representing their counterfactual advantage. In Equation 3.1, τ represents the agent's action-observation history and is a function of O and A, as referenced in Section 2.3. Furthermore, in this equation, u^i represents each potential action the agent could take, and this determines the difference for the agent taking some other action (the counterfactual baseline), with $Q(s, \boldsymbol{u})$ representing the Q-value for the joint-action which includes the actual action that the agent took. The default action is a design choice left up to the user. A visualisation of the flow of information is given in Figure 3.1, as it is one of the common ways to integrate a centralised critic, which is used in various other methods.

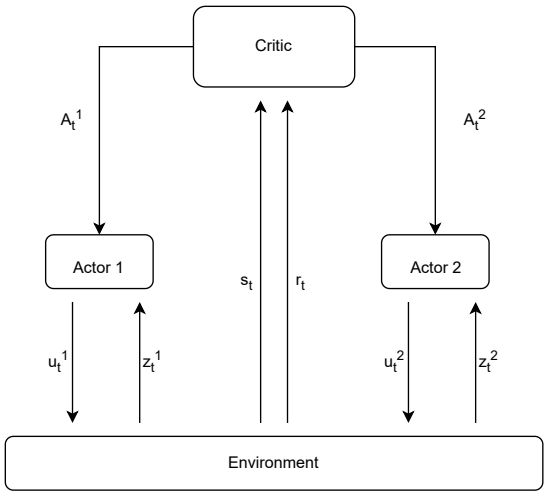


Figure 3.1: An illustration of the flow of information between the centralised critic and the decentralised actors. Here u_t represents the action taken by the agent, z_t represents the observation the agent receives, A_t represents the advantage estimate from the critic and t represents the current time step. Notably, this illustrates a simplified version of COMA's diagram where the true state space s_t and reward from the environment r_t is also given to the critic. Lastly, the superscript represents which agent this information is for.

Lastly, it is essential to note that COMA integrates a memory layer in the network [Foerster *et al.* 2018]. The need for a memory layer comes from the need to condition

on more than just the local observations; hence a recurrent neural network (which provides memory) can be used to condition the agent's behaviour on the entire history [Hausknecht and Stone 2015]. The choice of this memory unit can be flexible. However, it is common to use a gated model such as a *recurrent neural network* (RNN) [Rumelhart *et al.* 1986], *long short term-memory* (LSTM) [Hochreiter and Schmidhuber 1997] or a *gated recurrent unit* (GRU) [Cho *et al.* 2014]. Whilst COMA solves the issue of credit assignment with a centralised critic, it has some drawbacks that are problematic. Namely, this method has to have a centralised critic to estimate the counterfactual value efficiently. COMA also requires a simulator to train, which is problematic as we may not always be afforded said luxury [Zhang *et al.* 2021]. Finally, because COMA needs to evaluate the joint action values, this does make it more challenging to train, explore and coordinate [Foerster *et al.* 2018].

Whilst COMA assists agents in credit assignment in cooperative MARL another prominent method for solving the same task is a *value decomposition network* (VDN) [Sunehag *et al.* 2017]. The VDN helps to decompose the joint action-value function into individual value functions for each agent so that the team's reward is the sum of all individual agent action-values. This value decomposition is essential, as it leads to better overall performance than fully centralised and independent agents, and it is a modular method [Sunehag *et al.* 2017]. Here, modular refers to how this is a component that can be used in conjunction with another method. This decomposition can be represented using the following equation:

$$Q((\tau^1, \tau^2, \ldots, \tau^n), (a^1, a^2, \ldots, a^n)) = \sum_{i=1}^{d} \tilde{Q}_i(\tau^i, a^i), \tag{3.2}$$

where i represents a specific agent's index and d is the number of agents. Here, a memory layer (LSTM in this particular case) stores additional information from historical observations [Sunehag *et al.* 2017]. A key result of the VDN is that it can overcome the "lazy agent" problem, which is when one agent has a beneficial policy which discourages another agent from learning through exploration because the exploration may be detrimental to the first agent and hence result in a worse overall team reward [Sunehag *et al.* 2017]. Whilst the VDN method achieves high performance in various cooperative tasks, two underlying issues make it not feasible for our specific use case. Firstly, the VDN requires that we have centralisation to learn each agent's action-value function; in our circumstance this could cause the method to not be scalable and hence, is problematic [Papoudakis *et al.* 2019]. Secondly, their approach uses weight sharing for specific networks, and it is found that without this weight sharing, the lazy agent problem can reemerge [Sunehag *et al.* 2017]. The primary issue with weight sharing is that it causes the agents to become increasingly centralised, with full weight sharing (also referred to as parameter sharing), where all policies are represented using a single neural network [Terry *et al.* 2020b].

Action Decoding

Two well-known algorithms that achieve high performance in the cooperative case are those of the *Bayesian action decoder* (BAD) and *simplified action decoder* (SAD) [Foer-

ster *et al.* 2019; Hu and Foerster 2019]. BAD targets the partially observable setting in MARL, where agents cannot access all information. BAD attempts to utilise the public belief of Nayyar *et al.* [2013]; however, it seeks to scale it to large state-spaces [Foerster *et al.* 2019]. Here, public belief utilises knowledge common to all agents to infer their private information. This conditioning process on public features enables agents to consider the potential actions of their counterparts, leveraging the shared information among all agents. As a result, agents can adapt their actions more effectively in response to the anticipated behaviours of others. This consideration is done for each agent; importantly, every agent can compute this independently, and they will come to the same result. The reason why each agent arrives at the same result is that they all condition their private belief on the same public belief, which captures the collective knowledge of all agents about the environment and is accessible to everyone. BAD takes the idea of public belief and repurposes it as a third-party agent. The third-party agent utilises all the observations available to all agents and the public belief to generate a policy [Foerster *et al.* 2019]. To this end, it performs a deterministic mapping of an agent's private observation to an action only they can access. In particular, this is achieved by letting the central third-party agent select a policy for an agent based on the belief and the public features. Formally, the public belief at time t is represented by $\mathcal{B}_t$, where $\mathcal{B}_t = P(f_t^{private} | f_{\leq t}^{public})$ with $f^{private}$ representing the private information of a given agent and this is conditioned on all public features that have been seen up until that point as represented by $f_{\leq t}^{public}$. The exact public belief update for an agent is given by the following equation:

$$P(f_t^i | u_t^i, \mathcal{B}_t, f_t^{public}, \overset{\triangle}{\pi}) = \frac{P(u_t^i | f_t^i, \overset{\triangle}{\pi}) P(f_t^i, \mathcal{B}_t, f_t^{public})}{P(u_t^i | \mathcal{B}_t, f_t^{public}, \overset{\triangle}{\pi})}, \tag{3.3}$$

where u_t^i represents the action taken by an agent at a given time step t, $\overset{\triangle}{\pi}$ represents a partial policy which maps an agent's private observation (represented here as f_t^i) to an action. Although Equation 3.3 provides an exact value for the *public belief update*, in practice, this is not feasible except when the state space is small, and hence an approximation for it is used instead [Foerster *et al.* 2019]. Whilst BAD is capable of emergent communication protocols (rules which allow the agents to share information with each other), the algorithm requires handcrafted features as it requires expert knowledge of the system's dynamics [Hu and Foerster 2019].

Furthermore, using this third-party agent, which directly controls all other agents, results in this algorithm being highly centralised as it is incapable of functioning without the central agent. Lastly, one of the issues with BAD is its poor computational complexity, hence, the creation of SAD.

The algorithm also addresses one of the weaknesses with Bayesian methods, namely that they require expert domain knowledge to construct valuable features. Bayesian methods are generally not capable of computing exact updates in MARL given the complexity of certain circumstances [Strens 2000; Duan *et al.* 2016]. A potential circumstance where Bayesian methods may not be viable are when the domains are complex, such as high dimensional state spaces like visual input, in which case function approximators may be more desirable. One of the ways this is addressed is by incorporating

domain-specific knowledge which alleviates some of its poor computational complexity [Duan *et al.* 2016]. The incorporation of domain-specific knowledge to address the limitations of Bayesian methods presents a challenge as it requires handcrafting features, which is not always straightforward. In highly complex or nuanced settings, such as autonomous driving, determining the necessary features can be difficult, and the visual input is often complex. Therefore, this algorithm cannot be readily applied in these cases. SAD addresses some of BAD's fundamental flaws, such as the computational complexity of tracking the public belief [Hu and Foerster 2019]. The SAD algorithm achieves similar goals to BAD via an alternative approach involving information sharing between agents [Hu and Foerster 2019]. In particular, SAD lets agents take two actions at each time step, one greedy action and the other an action as determined by its policy [Hu and Foerster 2019]. The greedy action is deemed best by the agent's policy and is perceived to have the greatest estimated Q-value at the current time step [Sutton and Barto 2018]. The second action is considered to be the exploratory action [Sutton and Barto 2018], which is executed by the environment [Hu and Foerster 2019]. SAD simplifies the computational complexity of public belief by using memory (in the form of RNNs) to implicitly learn the distribution of Markov states given the history of action-observations [Hu and Foerster 2019]. Another component of SAD is that it uses a recurrent DQN with a *prioritised distributed experience replay buffer* inspired by Schaul *et al.* [2015], Horgan *et al.* [2018] and Kapturowski *et al.* [2019]. The distributed nature of the buffer means that the data is generated by multiple working agents, which are actors that run the agent's policy in each of the actors' own environments [Horgan *et al.* 2018]. The prioritised attribute refers to how the learning agent, which updates the actors' policies, samples the most valuable data more often, hence giving that data priority [Horgan *et al.* 2018]. Lastly, SAD uses a VDN to decompose the joint action-value function into individual action-value functions for each agent, which allows for credit assignment for each agent involved. However, despite SAD's improvement upon BAD, some issues still prevent solving our specific problem. In particular, SAD relies on a centralised critic during training to perform two crucial functions. These are the sharing of the greedy action, the agent's chosen action, and the VDN. In a fully decentralised setting, it may be possible that agents receive different rewards whilst still cooperating. In such a scenario, unless the agent could communicate with us at each time step what their reward and local observation was, it would not be feasible to use a VDN. This issue arises from the fact that VDN still requires this information and would need to communicate information to account for the lack of centralisation. Since communication itself is somewhat prohibitive, scenarios exist where we would not be able to share this information due to bandwidth constraints. Lastly, the experience replay method has stability issues in MARL as the environment becomes non-stationary from each agent's perspective as agent training progresses [Lowe *et al.* 2017].

3.3.2 Fully-decentralised Cooperation

Finally, one potential algorithm which alleviates the requirement of centralisation is proposed by Jaques *et al.* [2019]. Here Jaques *et al.* [2019] have their rewards augmented, similar to work based on curiosity-driven learning [Schmidhuber 2010; Pathak *et al.* 2017]. In curiosity-driven learning, agents typically model their environment us-

ing modules dedicated to learning the dynamics (what effect an action will have on the environment) and the inverse dynamics (what action did the agent take to change the observation from z_t to z_{t+1}) [Pathak *et al.* 2017]. This helps the agent explore areas where the modules' predictions are incorrect and hence, potentially not fully explored. This is focused on learning what the agent can directly control [Pathak *et al.* 2017]. Instead, Jaques *et al.* [2019] attempt to mimic *social learning*, which many consider one of the reasons humans can coordinate and operate at a large scale [Herrmann *et al.* 2007; Van Schaik and Burkart 2011; Harari 2014; Fernandes and BF 2017]. The fundamental idea in social learning relates to how humans do not only learn from their own actions; instead, we learn by observing others [Bandura 1977]. Furthermore, this method is more applicable to novel settings compared to prior works [Sequeira *et al.* 2011; Peysakhovich and Lerer 2017; Hughes *et al.* 2018; Jaques *et al.* 2019] due to its use of learned features rather than hand-crafted ones. An essential part of social influence is providing agents with a reward for having causal influence on another agent. Here, causal influence refers to how one agent impacted another agent's behaviour, and it is determined using counterfactual reasoning, the same concept that COMA utilises [Foerster *et al.* 2018; Jaques *et al.* 2019]. Counterfactual reasoning instead is computed by simulating what effect the agent would have by taking other actions at that time step instead of the actual chosen action [Jaques *et al.* 2019]. This influence is rewarded at each time step. Influence is an additional reward to the environment's reward at each time step. The weight for each of these rewards is a hyperparameter. The causal reward for an agent at a given time step t is given by:

$$c_t^i = \sum_{j=0, j\neq i}^{N} KL\big(p(a_t^j|a_t^i, z_t^i) \| p(a_t^j|z_t^j)\big), \tag{3.4}$$

where KL represents the KL divergence, j and i represent two separate agents. It is noteworthy that other metrics that measure divergence are applicable [Jaques *et al.* 2019].

Here Jaques *et al.* [2019] achieve notably high performance in Sequential Social Dilemmas (SSDs) [Leibo *et al.* 2017]; these are multi-agent games that are partially observable as well as spatially and temporally extended. In these SSDs, Jaques *et al.* [2019]'s agents' performance is noteworthy as they face a challenging cooperative task where they can choose to defect and act greedily over the short term, giving them immediate rewards [Jaques *et al.* 2019]. However, if all agents do cooperate, then the overall reward for each individual agent will be higher [Jaques *et al.* 2019].

Whilst in Equation 3.4, the counterfactual reasoning requires us to know how the target agent (the one whose response we are modelling) would react to each of our potential actions given the current observation, this introduces a small issue. Namely, that to accurately know these values, we would need centralisation. However, Jaques *et al.* [2019] propose to overcome this by using an internal module that models other agents using a neural network with LSTM layers. This internal module can instead calculate the counterfactual, overcoming the need for centralisation. The module fits into a subset of work surrounding the *"Machine" Theory of the Mind* [Botvinick *et al.* 2017; Rabinowitz *et al.* 2018]. One of the seminal works in the original Theory of

Mind was a work in cognitive sciences, particularly in Chimpanzees by Premack and Woodruff [1978]. The fundamental idea in this work is whether animals (specifically Chimpanzees) can reason about complex systems, and make predictions about the behaviour of others. In this case, it refers to the ability to reason over states that are not directly observable and leverage this to make predictions about the behaviours of others [Ng *et al.* 2000; Baker *et al.* 2011 2017]. The Machine Theory of Mind refers to agents performing a similar task [Rabinowitz *et al.* 2018]. Here, work such as that of Rabinowitz *et al.* [2018] attempt to do so by having an observer learn how to autonomously model other agents and their behaviours using limited data.

Despite Jaques *et al.* [2019]'s approach addressing the need for decentralised techniques and methods which are not hand-crafted there is a critical problem. Namely, we may have scenarios where there are multiple agents we wish to influence, and we may find that our agent focuses more on influencing others than solving the task at hand. Whilst this issue can be remedied by lowering the coefficient for the causal influence reward it may be that our agents do need to influence others. However, which agent to influence and how to exclusively target them is a significant issue. Furthermore, this approach only works for cooperative settings as we would not want to be rewarded for helping to influence our opponent's actions in a competitive setting in case this may benefit competitor agents. This aversion to rewarding influence is similar to curiosity-driven learning, the additional reward is an auxiliary reward and not necessarily the goal we have in our environment [Pathak *et al.* 2017]. Therefore, due to this high degree of modelling and the need to acknowledge the competitive case, we seek other methods to solve our problem.

3.4 General-sum Algorithms and Competitive Case

We have combined the work on general-sum and competitive algorithms into a single section, as some general-sum approaches may lead agents to cooperate if it aligns with their own interests. However, this is not the same as unconditional cooperation, which can be observed in specific settings like those in the Sequential Social Dilemma paper by Jaques *et al.* [2019] or Foerster *et al.* [2019]'s BAD and Hu and Foerster [2019]'s SAD methods. Unconditional cooperation in MARL are scenarios where agents cooperate regardless of their own self-interest. This cooperation can be achieved through explicit cooperation mechanisms, such as influence rewards, or by using cooperative-focused algorithms, like BAD and SAD.

3.4.1 Centralised Critic

One algorithm that addresses some prior concerns in Section 3.3 is that of Lowe *et al.* [2017]. They develop a method which relies solely on local agent observations and do not presume any communication between agents. An additional benefit is that it can operate in cooperative, competitive and mixed-settings. One key component in their work is that of the *centralised action-value function*, similar to Equation 3.2. This function is represented by $Q_i^{\boldsymbol{\pi}}(\boldsymbol{x}, a^1, a^2, \ldots, a^i, \ldots, a^n)$, where i represents which agent this value is for and $\boldsymbol{x}$ is the state information, which in its simplest case is the set of all

agent observations. Additionally, it takes into account each agent's action as indicated by a. Each agent uses this to optimise their objective function. An advantage of this approach is that each agent learns a separate centralised action-value function; hence agents can have varying reward structures, which may allow for competition [Lowe *et al.* 2017]. However, a disadvantage of this method is that it is fundamentally designed with a CTDE framework in mind as it allows for an augmented critic with access to extra information about other agents' policies during training. It is possible to relax this constraint, but it requires explicit modelling of other participating agents' parameters. This approach would be challenging and unreliable in our problem given the consistently mentioned non-stationarity involved in MARL [Lowe *et al.* 2017; Sunehag *et al.* 2017; Foerster *et al.* 2017a], which means the agent would condition its policy on a constantly moving target should the other agent be learning.

3.4.2 Awareness of Others

Alternatively, we have a set of algorithms based on the *learning with opponent awareness* (LOLA) framework [Foerster *et al.* 2017b; Willi *et al.* 2022; Zhao *et al.* 2022]. The original LOLA algorithm integrates similar ideas to Jaques *et al.* [2019] but does so more directly. Here, the LOLA agent attempts to influence the other agent, however its goal is to shape that agent's learning. This shaping is done via an impact term in its update and is provided below:

$$\begin{aligned} &\boldsymbol{\theta}^1_{t+1} = \boldsymbol{\theta}^1_t + f^1_{LOLA}(\boldsymbol{\theta}^1_t, \boldsymbol{\theta}^2_t), \\ &f^1_{LOLA}(\boldsymbol{\theta}^1_t, \boldsymbol{\theta}^2_t) = \nabla_{\theta^1_t} V^1(\boldsymbol{\theta^1_t}, \boldsymbol{\theta^2_t}) \cdot \delta + (\nabla_{\theta^2_t} V^1(\boldsymbol{\theta^1_t}, \boldsymbol{\theta^2_t}))^T \nabla_{\boldsymbol{\theta^1_t}} \nabla_{\boldsymbol{\theta^2_t}} V^2(\boldsymbol{\theta^1_t}, \boldsymbol{\theta^2_t}) \cdot \delta\eta. \end{aligned} \tag{3.5}$$

Here $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$ represent agent one and agent two's respective parameters, V^1 and V^2 represents the total discounted reward for each agent as a function of their parameters, t represents the time step, δ is the step size for the first order update to this equation, and η is the second order update to this equation. The second term in this function represents the impact of agent one's anticipation on agent two's learning, enabling it to account for the effect of its actions on the second agent's learning. One of the most significant benefits of this algorithm is that it helps shape the learning of other agents in the environment by anticipating the other agent's learning. This is important as it does not assume that the other agents are just static components of the environment [Hernandez-Leal *et al.* 2017; Foerster *et al.* 2017b]. Hence, LOLA, unlike other methods, such as those of Zhang and Lesser [2010] does not attempt to learn the best response to other agents based on their updated policy parameters [Foerster *et al.* 2017b]. Instead, LOLA attempts to shape the policy update of these other agents with the explicit goal of maximising its own reward [Foerster *et al.* 2017b]. Therefore, it is fair to say that LOLA agents are self-interested and will learn to be cooperative if it benefits them.

A strong showcase of the effectiveness of MARL comes from the iterated Prisoner's Dilemma, which is represented in Table 3.1. Each agent can cooperate or defect in this game, as represented by C and D, respectively. For this simple game, depending on the combination of actions each agent took, we ended up at position CC, CD, DC or DD, where the first letter represents the row and the second represents the column. An

	C	D
C	(-1,-1)	(-3,0)
D	(0,-3)	(-2,-2)

Table 3.1: This table is a visual representation of Prisoner's Dilemma. Here C represents cooperate, D represents defect and each agent can choose one of these two options. The first reward for each element in the table is for agent one and the second reward is for agent two.

important concept of Nash-Equilibria defines the optimal strategy each agent should have [Nash Jr 1950]. In the iterated Prisoner's Dilemma, there are infinitely many Nash-Equilibria according to Folk theorem [Myerson 1997]; however, two particular ones are DD and "tit-for-tat". Tit-for-tat refers to how each agent will initially cooperate. Following this, the agents will repeat the other agent's previous action. For example, if the agent took a cooperative action (C) in the previous round, then in the following round, a tit-for-tat agent would also be cooperative. In this case, LOLA is capable of learning tit-for-tat and cooperative strategies resulting in the best reward from the set of evaluated approaches in this setting [Foerster *et al.* 2017b].

Despite the benefits of LOLA, there is one potential issue with its update in Equation 3.5. This equation assumes that the agent has access to the other agent's exact parameters. To remedy this limitation, Foerster *et al.* [2017b] do something similar to behavioural cloning [Bowling and Veloso 2002; Ross *et al.* 2010]. Behaviour cloning refers to mapping a set of situations (in our case states) to actions where the goal is to mimic another agent's behaviour, for example, when playing a game what actions would a human player take in the current state [Bowling and Veloso 2002; Kanervisto *et al.* 2020]. In this case, Foerster *et al.* [2017b] attempt to approximate the other agent's parameters using Equation 3.6. This instead can be used in Equation 3.5 as a substitute for the agent's exact parameters.

$$\hat{\boldsymbol{\theta}}^2 = \arg\max_{\boldsymbol{\theta^2}} \sum_t \log \pi^2_{\hat{\boldsymbol{\theta}}}(\boldsymbol{u}^2_t | s_t). \tag{3.6}$$

Here agent one is approximating the parameters of agent two (represented by $\hat{\boldsymbol{\theta}}^2$) and $\boldsymbol{u}^2_t$ is the joint action vector for the 2^{nd} agent.

However, there are issues with this algorithm that *proximal learning with opponent-learning* (POLA) and *consistent learning with opponent-learning awareness* (COLA) attempt to address [Willi *et al.* 2022; Zhao *et al.* 2022]. LOLA does not assume other agents may be modelling the LOLA agent similarly to how LOLA models them [Willi *et al.* 2022]. Therefore, the other agent may also be able to influence LOLA, which may create additional complexity as multi-level reasoning would be required to deal with how agents may continue to model each other. Willi *et al.* [2022] state that there is still some non-stationarity with LOLA as it is not able to achieve stable fixed points. Similarly, COLA cannot achieve stable fixed points either. Whilst COLA does propose an alternative update function to LOLA, it assumes access to some key information, such as the other agents' rewards, parameters and gradients. Furthermore, Zhao *et al.* [2022] note other critical concerns with LOLA, namely that LOLA often fails to learn its

intended policy when it is parameterised using complex neural networks. Additionally, Zhao *et al.* [2022] state that LOLA is not ideal in circumstances where it cannot directly access other agents' parameters. One concern with methods that actively model other agents with extreme depth is that the task will become increasingly complex as the number of agents increases because the system will need to do complex multi-objective optimisation. This optimisation may not necessarily be trivial as all agents in the system learn and respond to each other. Still, the underlying optimisation problem can be problematic, especially given the non-stationarity common in MARL. Therefore, approaches that make fewer assumptions and perform minimal modelling will be desirable and more widely applicable.

3.4.3 Fully Decentralised General-Sum Algorithms

One paper that deals with the same problem as we do is from Albrecht and Ramamoorthy [2015]. Here a method for *ad hoc* coordination is proposed that aims for optimal flexibility and efficiency. Ad hoc coordination is the idea that the agent must be autonomous and requires no prior coordination. Here this means that the behaviour and type of agents that an agent is paired with is not known prior to execution. Firstly, there is no prior agreement in communication protocols. In our fully decentralised approach also focuses on the idea of no explicit communication. However, this method primarily relies on modelling using Bayesian techniques. This modelling is seen through the agent determining whether another agent is a specific type, given its history. This approach relies on Bayesian approaches. Thus, it would require that we can accurately model the problem. The modelling can focus on various components, such as the environment or a distribution over agent types or other agent's parameters (such as their neural networks' parameters) [Albrecht and Ramamoorthy 2015; Foerster *et al.* 2017b 2019]. Bayesian approaches allow us to create detailed and carefully tuned pre-trained models for a given task. However, the approach is limiting in some regards, specifically that we must know the potential prior distributions of agents and specific actions or potential state spaces. This approach can be quite limiting as it would have a highly detailed and large state-space in which tabular methods are not generally applicable. Nevertheless, in reinforcement learning, we can leverage deep learning techniques. Consequently, relying on prior information (or expert domain knowledge of a given problem) might constrain the method's applicability, particularly when addressing niche problems that lack well-established or existing models. For instance, consider a scenario where agents must collaborate to manage an unknown ecosystem, involving the interactions of numerous species and environmental factors. In this case, traditional models and domain knowledge might not adequately capture the complexities and intricacies of the ecosystem, limiting the effectiveness of the Bayesian approach. Adapting to such novel problems requires flexible algorithms that can learn and evolve without relying on pre-established models or prior assumptions. In cases where we have novel problems or tasks, a current method will not exist to model the problem or impart expert information. Lastly, using handcrafted features can be limiting as we must know what is essential beforehand to model the problem and use this technique accurately.

Therefore, while the method is desirable and applicable, there are many circumstances where it is impractical or prohibitive. Deep learning allows methods in more complex spaces. It does not require us to explicitly model the problem or pick handcrafted features. Although these methods are computationally expensive, the benefit is that deep learning techniques allow for automatic feature extraction and feature discovery in domains for which a Bayesian model does not currently exist or is challenging to accurately model. Hence, we draw great inspiration from this method. However, we wish to expand upon its ideas in a deep multi-agent learning system.

3.5 Model Free RL

Sharing Parameters

Prior work examines how PPO can be improved in a multi-agent setting via simple modifications and fine-tuning without the need to completely redesign the entire algorithm [Terry *et al.* 2022; Yu *et al.* 2021]. Multi-Agent PPO (MAPPO) is a standard augmentation of PPO that adapts the algorithm to the multi-agent setting [Yu *et al.* 2021]. Gupta *et al.* [2017] notes that parameter sharing allows all agents to have the same parameters for a given policy. This approach allows the policy to use the experience of all agents concurrently. Currently, non-stationarity is a crucial concern in MARL and centralised architectures are one way to address this problem [Papoudakis *et al.* 2019]. Some methods such as those of Gupta *et al.* [2017] and Chu and Ye [2017] utilise this technique in cooperative tasks. However, Terry *et al.* [2022] notes that full parameter sharing is considered an extreme case of centralisation. Full parameter sharing implies that all agents share a single network for their policy function [Terry *et al.* 2022]. Terry *et al.* [2022] illustrates how it helps to improve performance without necessarily changing the underlying algorithm. Sharing can be achieved by using a single critic alongside one or multiple actor networks.

Modifying PPO

Coordinated PPO (CoPPO) Wu *et al.* [2021] builds upon MAPPO and extends it using adaptive step sizes. This modification can allow the algorithm to outperform certain standard MARL benchmarks. Lastly, Terry *et al.* [2020b] illustrates that parameter sharing is a successful approach in environments where agents are not identical and differ regarding their action space, observation space, or both. However, all the above approaches require some centralisation, often during training (such as the CTDE methods) or throughout the entire process. Despite this, centralisation may not always be possible in real-world scenarios; fully-decentralised methods may be preferable in certain settings [Zhang *et al.* 2021]. Furthermore, the fully-centralised case also suffers from scalability issues when we have many agents interacting [Gupta *et al.* 2017; Hernandez-Leal *et al.* 2019]. We instead aim to improve the performance of fully-decentralised *PPO* in fully-cooperative MARL.

3.6 Entropy Regularisation

Current Use of Entropy

Entropy regularisation (ER) is a commonly used method in RL, which assists agents in their exploration process [Schulman *et al.* 2017ba; Ahmed *et al.* 2019]. By adding an entropy term to the objective function, ER prevents agents from becoming deterministic prematurely and encourages a more diverse set of actions. This is important because exploration is crucial for finding good, or potentially optimal policies. If the set of sampled state-action pairs is not diverse enough, the agent may converge to a suboptimal policy [Ahmed *et al.* 2019]. The emphasis on diverse actions is particularly pertinent in the realm of Deep Reinforcement Learning. Here, the reliance on gradient-based optimization means we can't simply direct the agent to select a particular action, as doing so would deprive us of meaningful gradients. The lack of these gradients would impede the agent's learning, making techniques like ER essential.

Ahmed *et al.* [2019] illustrate that in stochastic environments, ER can create better objective functions to optimise, leading to higher entropy policies that enable a more rapid learning process and potentially better final rewards. Furthermore, ER helps create more stochastic policies, which generally have smoother objective landscapes and are beneficial in policy optimisation [Ahmed *et al.* 2019; Zhou *et al.* 2020]. The main definition of entropy used when performing ER is Shannon entropy [Shannon 1948], denoted as:

$$H(\pi(a^i|z^i)) = -\sum_d p(a^i_d)\log(p(a^i_d)), \tag{3.7}$$

where a^i_d represents an action that agent i can take, with $p(a^i_d)$ representing its corresponding probability, and summed over all possible actions. π represents the policy of the agent in a given state z. Generally, this term has a coefficient represented using λ or β, which weighs the importance of ER in the objective function [Mnih *et al.* 2016; Lowe *et al.* 2017; Zhou *et al.* 2020].

Modifying Entropy

The original outline of this approach was introduced by Williams and Peng [1991] in the form of entropy maximisation. The goal of entropy maximisation was to prevent the agent from converging to a single chosen output [Williams and Peng 1991]. This refers to the case when the agent selects a single action for a given state continuously, before taking exploratory actions that may be beneficial. Thus, it prevents the agent from becoming deterministic early on, and during the training of the agent. Preventing determinism is often desirable for reasons outlined at the start of Section 3.6. However, one effect of this entropy is that the agent's network sacrifices some performance to achieve greater entropy by continuing its exploration. It has continued to see use in modern RL due to the desirability of encouraging agents to explore.

A common method that typically uses standard Shannon entropy is that of maximum entropy RL (MaxEnt RL). Others have used this approach, and a common reason for it is the robustness that it can allow for [Ziebart *et al.* 2008; Toussaint 2009; Haarnoja

et al. 2018]. This approach may help improve exploration and robustness for agents, as indicated by Ziebart [2010] and Haarnoja *et al.* [2017] showing that this approach may be robust to model estimation errors. Here, model estimation error refers to the discrepancies between the model used by an agent and the real environment which it functions in, and this can degrade agent performance when the estimation error is high. We have one crucial concern, namely, the diversity of their actions themselves may be too noisy for MARL. As stated by Haarnoja *et al.* [2018], the agents aim to maximise their expected rewards and *their entropy*, aiming to succeed at a task whilst acting as *randomly as possible*. This is not necessarily negative in single-agent RL, as agents are in essence, encouraged to explore more widely but not to continue in suboptimal areas [Haarnoja *et al.* 2018]. This approach has been incorporated into MARL by authors such as Xing *et al.* [2021] who use a similar concept to generate teammates with a wide range of behaviours and varying degrees of stochasticity in their policy. Whilst Xing *et al.* [2021] motivate the use of this method for generating teammates to pre-train with, our focus is instead on having agents learn simultaneously in the same environment. However, in our case, the goal of MaxEnt RL may not be suitable in fully-decentralised MARL. In fully-decentralised MARL, we already suffer from non-stationarity. Whilst our agents may already be selecting a diverse set of actions due to the encouragement from entropy, this entropy, though rewarding for a broader range of actions, also complicates coordination. This is because agents are not always incentivised to choose the most optimal action. Therefore, this motivates that augmenting the role of entropy or modifying the ER function may be desirable. In our particular case, this would be less diverse and possibly "random" actions with more certain and "reliable" actions. In this context, "random" refers to actions predominantly driven by the incentive to explore a broader variety of actions. In contrast, "reliable" denotes actions where probabilities are more focused around specific, consistent choices, leading to a less diverse action set.

One prominent algorithm which leverages a modified ER function is *learning implicit credit assignment* (*LICA*) [Zhou *et al.* 2020]. This method uses a centralised training regime and performs well in fully cooperative settings. Importantly, *LICA* uses an alternative entropy regularisation term called *adaptive entropy loss* (*ADE*) [Zhou *et al.* 2020]. The *ADE* function is formulated as:

$$\text{ADE} = -\xi \cdot \frac{log p_d^j + 1}{H(p^j)}. \tag{3.8}$$

Here, j is the agent, and d represents the action taken; thus p^j is the set of probabilities of each action that agent j can take with p_d^j being the probability agent j takes action d. Lastly, ξ represents the coefficient of the entropy term. Generally, the objective function for the agent is given by:

$$J(\theta) = \mathbf{E}[Q_\phi^\pi(s_t, u_t)] + \xi \mathbf{S}, \tag{3.9}$$

where $Q_\phi^\pi(s_t, u_t)$ represents the joint value function for the agents and is equal to the expected return of the discounted cumulative rewards over the finite time horizon. Furthermore, $\mathbf{S}$ represents the entropy term in a generalised form, given that there exist multiple methods to measure entropy. In this equation t represents the current time step of the agent, u_t represents the set of actions the agents took and θ_j represents

the parameters of agent j's policy network. In the second term, $\mathbf{S}$ in this equation is for ER and could be based on Shannon entropy, ADE or a custom approach. This entropy term can be weighted with coefficient ξ that can be fine-tuned.

Building on our previous discussion of how entropy influences the policy and action selection of an agent, we now aim to explore its repurposing in a fully-decentralised setting. Whilst many methods integrate or use entropy to encourage exploration, and hence, prevent agents from becoming too deterministic, we propose an alternative use of entropy. Specifically, if we discourage entropy by penalising it rather than rewarding it, we could encourage agents to be more deterministic. This alternative of a more deterministic agent may be undesirable when agents have complete control of an environment, such as when one central controller controls all individual agents, or if a single agent exists and is the only learning agent in an environment. However, when there are multiple agents also learning and acting independently, this encouragement of a more deterministic, and hence, more "reliable" agent may be desirable. Therefore, we propose to re-examine the role entropy could have for an agent in fully-decentralised MARL. We provide greater detail and our reasoning for this approach in Chapter 4.

3.7 Summary

In conclusion, we have reviewed various methods for MARL. While some exist that can operate fully-decentralised, they often require exchanging information or modelling opponents. We have identified a gap in the literature for methods that can perform in fully-decentralised MARL while relying on something other than opponent modelling. To address this gap, we propose modifying the role of entropy and leveraging non-stationarity to our advantage, allowing each agent to act independently without requiring centralised information exchange. We believe this approach is novel and has the potential to yield significant benefits. In the next chapter, we explain our reasoning in-depth and describe our proposed method.

Chapter 4

Methodology

Considering that adjusting entropy levels can affect agent behaviour, we propose a method that incorporates entropy for an entirely different purpose. It should be emphasised that entropy adjustments can be applied individually to each agent, enabling modifications to their learning behaviour, which can change the way agents are encouraged to act during learning to provide outcomes that might be more desirable than promoting individual exploration. As a result, the method we present can be implemented separately for each agent during training, eliminating the need for additional modelling of other agents in the system. Furthermore, since our approach uses neural networks to perform function approximation, it can be effectively applied to various tasks with high-dimensional state spaces.

We break down our methodology in the following manner. In Section 4.1, we provide an example of a case where having lower entropy may be desirable in a multi-agent setting. By lower entropy, we mean that exploration might not necessarily be carried out by individual agents as it is done in single-agent RL. Instead, each agent, by acting stochastically during the learning process, already induces some exploration, which, compounded by other agents doing the same, may already create sufficient exploration. Following this, in Section 4.2, we discuss the problem of non-stationarity, link entropy to this problem, and explain how this may help address this issue. Inspired by Shannon entropy, we introduce an alternative way of measuring entropy for our agents. Finally, in Section 4.3, we illustrate how we incorporate the various methods and provide their objective functions. We conclude this chapter by demonstrating the pseudocode for our approach.

4.1 Motivating Example

To illustrate why we may want agents to take less diverse actions, we consider the following practical example. Suppose someone is playing a football game, and there are only three minutes left before the match ends. Let us focus on one specific player who is currently in possession of the ball and is near but not close to the opposing team's goalpost. There is a slight chance they are able to score as the opposing team

changes their focus to them as they quickly converge on the player. The distance from them to the goalpost is not near enough for them to make the shot reliably. However, there is a teammate nearby. They are watching their teammate as they plan their next move. Their teammate is currently open, and they can kick the ball to them, scoring their team the winning point of the game. In this situation, this player can continue to try and score, which may be risky as all opposing players are currently surrounding them, or they can kick the ball to their friend. Here our agent is relying on their teammate to make a good decision and not to act randomly, as when they kick the ball, they hope their ally will receive it and then score. Figure 4.1 illustrates one of their essential concerns for making this decision. If their teammate was a reliable agent, then they would be more confident that they would take a specific action given the current circumstance. For example, they are more likely to stay still and receive the ball, or they are more likely to continue moving forward to the opposing team's goalposts. However, if they were an unreliable agent, their teammate may do something entirely unanticipated, such as turning around and returning to their team's goalpost. This example illustrates why we might want to have more certain agents. This certainty or reliability makes the agent more predictable; hence, in a cooperative setting, they are potentially a more helpful teammate in certain scenarios. Furthermore, as the agents become more certain about their actions, they may alleviate the non-stationarity.

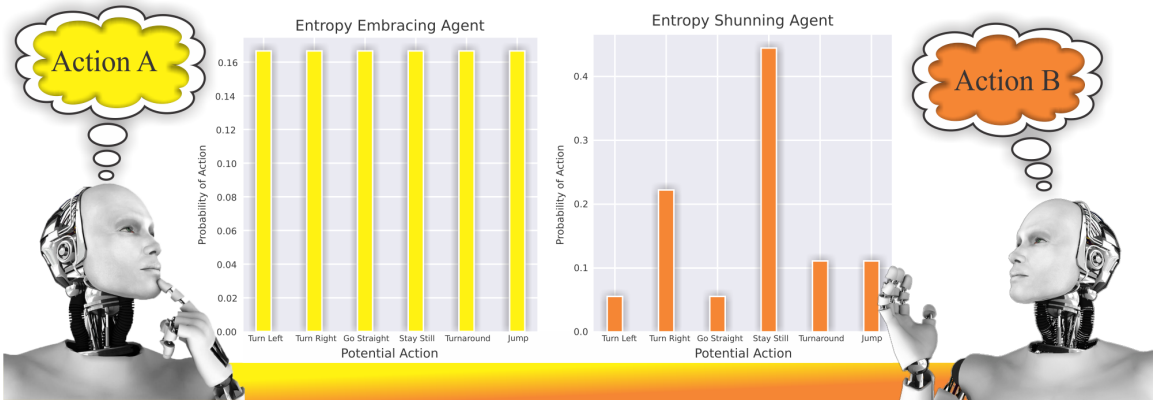


Figure 4.1: Here we have two agents. One agent is an entropy-shunning agent represented on the right, which attempts to become more certain. The other agent on the left is an entropy embracing agent, which seeks to take more diverse actions. The probability distributions in the background provide a visual example of how their current policy may appear for a given state.

4.2 Handling Non-Stationarity

4.2.1 Non-Stationarity and Entropy

It is well known that the MARL domain, particularly the fully-decentralised case, suffers from non-stationarity [Hernandez-Leal *et al.* 2017; Papoudakis *et al.* 2019; Terry *et al.* 2022]. Current entropy regularisation methods perform well in single-agent RL due to the increase in exploration. However, in MARL, an alternative approach may be more

suitable given the inherent non-stationarity of this setting. Typically, as Equation 3.9 illustrates, entropy is an additive term, with entropy encouraging exploration [Mnih *et al.* 2016; Schulman *et al.* 2017b; Lowe *et al.* 2017; Zhou *et al.* 2020; Terry *et al.* 2022 2020b; Yu *et al.* 2021]. Instead of encouraging exploration, which may lead to more random actions that could potentially make an agent's partner seem unreliable we propose to do something different. We incentivise agents to minimise entropy, which makes them more reliable as they seek to increase the likelihood of certain actions. This growth refers to how the agent is incentivised in its objective function to increase the probability of certain actions by decreasing the probability of others. This therefore makes these agents likely to be better teammates as they will not be as likely to take exploratory actions, which may exacerbate the non-stationarity. This makes these agents less susceptible to the inherent non-stationarity in MARL.

4.2.2 Repurposing Entropy

Traditionally, entropy regularisation is used to encourage agents to take a more diverse set of actions, especially when they control the environment. This diversity is beneficial as the cause and effect of actions become more apparent, whether the environment changes due to the agents' actions themselves or because of potential stochastic events that define the environment." However, when other agents replicate this behaviour in MARL, this may lead agents to be misguided about the cause and effect of their actions, as the outcome of their actions could be due to the joint action of multiple agents rather than their own individual action. Therefore, we seek to evaluate whether encouraging this behaviour or penalising agents for it (which is counter to the current approach) is beneficial.

We propose to use the term *entropy-embracing* to refer to agents that are rewarded for exploration, resulting in a more uniform distribution of actions. Alternative to these embracing agents are *entropy-shunning* agents. We provide a simple visual in Figure 4.1 as an illustration of these two approaches. These agents get penalised for exploration and seek to be more deterministic over time and hence, are more reliable. To do this, we subtract the entropy regularisation term (as seen in Equation 3.9) rather than add it. These entropy-shunning agents gradually become more reliable, as they receive a penalty proportional to the low probability of their chosen actions. This penalty encourages them to increase the probability of other actions, making their behaviour more predictable and consistent. Although there may be situations where this approach is less desirable than the alternative, we believe that fully-decentralised MARL environments already exhibit sufficient randomness to allow for our proposed approach. Our motivation for this approach is that (in contrast to many single agent RL environments) MARL environments already exhibit enough randomness. This is due to the nature of MARL, where the agents already sample from stochastic or continually adapting policies, which we argue are sufficient for exploration. Therefore, given the elevated levels of non-stationarity in MARL, reducing the entropy, which leads to slightly less stochastic agents may be a potentially beneficial alternative. We demonstrate our alternative approach using two methods, these are ADE(*neg*) and RECO(*neg*) which have the entropy regularisation term subtracted – as represented by the label (neg).

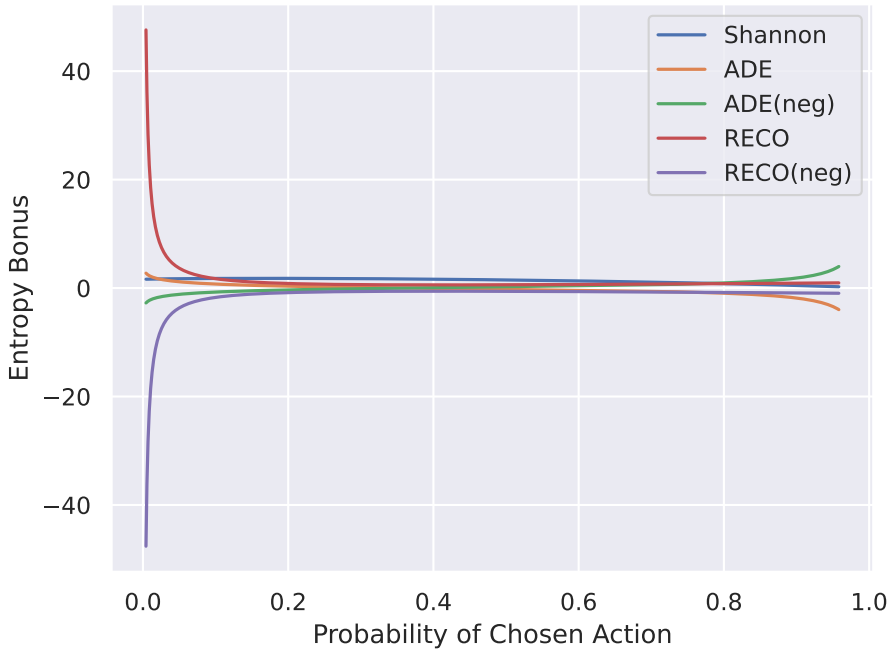


Figure 4.2: A visualisation of the effect the probability of an action has on the entropy for each of the ER approaches. Here the y-axis represents the reward the entropy regularisation adds to the reward at each time step should the agent take the action with the given probability, as shown on the x-axis.

4.2.3 An Alternative Measure

We introduce a simple and naive alternative called *reliability coercion* (RECO). RECO represents a simple augmentation that rewards agents for taking unlikely actions and would make the agent entropy-embracing. It is similar in purpose to Shannon entropy and *ADE* but calculated differently. We use this form to demonstrate a similar but alternative design approach to the current approach. The equation of RECO is given as:

$$\text{RECO}(\pi(a^i|z^i)) = \xi \cdot \frac{\sum_y p(a_y)^2}{p(a_d)}. \tag{4.1}$$

For agent i, the probability of a given action a_y is represented by $p(a_y)$, which is summed over all potential actions (as indexed by y) that the agent could take at a specific time step. The probability of the action that the agent actually took is represented by $p(a_d)$, and the coefficient ξ is used to weight the importance of this term.

We motivate the design of the simple RECO term using Figure 4.2 and a straightforward example. In this example, we make one action increasingly likely. Therefore, we begin by assigning a probability near zero (but not zero) to an action, and then gradually increase the probability of that action, while uniformly decreasing the probabilities of all other actions. We do this to demonstrate what happens should the agent take the most likely action or an exploration action. This graph assumes that an agent has access to six potential actions. Initially, all actions are equally likely. However, we wish

to increase the likelihood of a specific action denoted as d. We increase the likelihood of action d by decreasing the probability of every other action equally. In the example where one action is made increasingly likely, the reward of selecting that action as its probability increases does not significantly change after a certain point, as shown in Figure 4.2. This figure illustrates the effect of the probability of an action on the entropy for each of the entropy regularisation approaches. The y-axis represents the reward that the entropy regularisation adds to the reward at each time step if the agent takes the action with the given probability, as shown on the x-axis. However, should we choose one of the other less likely actions, we see a significant increase in the reward as the action becomes increasingly unlikely. Lastly, we create RECO(neg) by subtracting this reward instead of adding it. This subtraction creates a second entropy-shunning agent to demonstrate our alternative approach.

4.3 Modifying PPO

When modifying *PPO*, we modify the agents using the single case version outlined in the background, in Section 2.2.5. We do this as each of the agents are treated independently, as we focus on the fully-decentralised case. We make some alterations to all entropy (therefore this excludes the base approach) approaches when applying them to *PPO*. We restrict the range of the RECO-based and the ADE(neg) terms to prevent the network from encountering exploding gradients in its updates. In particular, this restriction is performed on the entropy generated during policy roll-out and policy evaluation. This technique, similar to Pytorch's clipping, ensures that the values do not go out of a specific range, and we refer to this as clamping. These modifications are essential as they prevent losses from causing the network to explode due to large updates. In particular, the modified components are the surrogate objective function ($L_t^{CLIP}(\theta)$) and the additional entropy term. Here, $L_t^{CLIP}(\theta)$ is modified by adding entropy from policy rollout to the log probabilities. This particular modification is detailed in Algorithm 2, and we denote this modified version using $L_t^*(\theta)$, however we explain what the necessary modifications are. We use an additional entropy term as outlined in Sections 2 and 3, which is added as the term Ω as the agent interacts with the environment (line 20 in Algorithm 2). We add Ω to the objective function in Equation 4.2. Additionally, this term is accumulated during the agents' training – indicated in line 7. This term serves as additional entropy and slightly modifies Equation 4.2.

$$L_t(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t)] \tag{4.2}$$

Furthermore, before this additional entropy gets accumulated we add the entropy term to the agent's log probabilities during policy rollout (line 8 in Algorithm 2). When providing the various approaches and their objective functions, we exclude a negative version Shannon entropy. This exclusion is because ADE as Zhou *et al.* [2020] state that the common form of entropy regularisation (Shannon entropy) is undesirable given its properties of its derivative. Instead, ADE achieves the same goals as Shannon entropy but allows for a more dynamic and robust, and it may maintain a smoother objective

landscape, which as Ahmed *et al.* [2019] demonstrate is a highly desirable property. Below we provide the objective function for each of our methods.

- Base PPO

$$L_t^{base}(\theta) = \hat{\mathbb{E}}_t[L_t(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t)] \tag{4.3}$$

- PPO using Shannon Entropy

$$L_t^{Shannon}(\theta) = \hat{\mathbb{E}}_t[L_t^*(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t) - \xi \cdot \sum_y p(a_y)\log(p(a_y))] \tag{4.4}$$

- PPO using ADE

$$L_t^{ADE}(\theta) = \hat{\mathbb{E}}_t[L_t^*(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t) - \xi \cdot \frac{logp_d + 1}{H(p)}] \tag{4.5}$$

- PPO using ADE(neg)

$$L_t^{ADE(neg)}(\theta) = \hat{\mathbb{E}}_t^*[L_t^*(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t) + clip(\xi \cdot \frac{logp_d + 1}{H(p)}, \kappa)] \tag{4.6}$$

- PPO using RECO

$$L_t^{RECO}(\theta) = \hat{\mathbb{E}}_t^*[L_t^*(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t) + clip(\xi \cdot \frac{\sum_y p(a_y)^2}{p(a_d)}, \kappa)] \tag{4.7}$$

- PPO using RECO(neg)

$$L_t^{RECO(neg)}(\theta) = \hat{\mathbb{E}}_t^*[L_t^*(\theta) - c_1(V_\theta(s_t) - V_t^{targ})^2 + c_2 S\{\pi_\theta\}(s_t) - clip(\xi \cdot \frac{\sum_y p(a_y)^2}{p(a_d)}, \kappa)] \tag{4.8}$$

Here, we illustrate which methods make use of a modified surrogate function by indicating it with red, and we indicate each additional loss function in blue. For each agent j, the probability of taking action a_y is represented by $p(a_y)$, where y indexes the set of potential actions available to the agent. Additionally, a_d represents the action that is actually chosen by the agent during that step, and hence $p(a_d)$ represents the probability of this action. During policy rollout (lines 3 and 4 in Algorithm 2), the agent stores the log probability of its actions as a vector for each parallel actor at each time step. In the case of our modified PPO algorithm, we modify this by adding Ω however, for the base PPO algorithm we do not add this. Furthermore, methods which require any clipping have $clip$ in their equation. How we store the running entropy, add entropy to the log probability buffer and clip the additional entropy, is illustrated clearly in Algorithm 2. The pseudocode outlines how our methods and baselines can be integrated into the PPO algorithm by specifying the necessary modifications and their location. To improve clarity, we have highlighted these components in colour.

Algorithm 2 Modified PPO Algorithm

Input: $\xi, num\_updates, num\_steps, batch\_size, minibatch\_size$

$\Omega = 0$
Initialise agent buffers with size of number of steps the agent will take during policy rollout
for update=1,2,...,$num\_updates$ **do**
  for step=0,1,...,$num\_steps - 1$ **do**
    for actor=0,1,...,$N - 1$ **do**
      Collect log probabilities of actions, entropy, value estimate and additional entropy (ζ) as policy π_θ runs in environment for T time steps
      Ω += $\xi \times \zeta$ # accumulate additional entropy
      $logprobs[step] \times \Omega$ # use this entropy in the surrogate objective function
      Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
  end for
  for epoch=0,1,...,$num\_epochs - 1$ **do**
    $base = \Omega$
    for i=0,1,...,$batch\_size$, with step size $minibatch\_size$ **do**
      $\Omega = base$
      newlogprob, $entropy$, newvalue, ζ = get value(agent's past actions) # collect default entropy
      calculate value loss
      $base = \zeta \times \xi$ # use updated base for objective function
      calculate policy loss (pg_loss) before optimising
      pg_loss += $base$ # additional entropy on top of the standard $entropy$
    end for
  end for
end for

4.4 Summary

In conclusion, we have introduced an alternative approach that deviates from the current standard by aiming to penalise and minimise entropy instead of promoting it, which could lead to agents taking unnecessary actions. The non-stationarity inherent in MARL (especially in fully-decentralised MARL) may serve as a different and potentially superior source of agent stochasticity. This approach enables us to adjust agents without making any additional assumptions, such as access to other agents' parameters or efforts to model them. The primary limitation of this method, as discussed in Chapter 5, is its potential efficacy only in situations with dense rewards, as the non-stationarity might not suffice for exploration when rewards are sparse. In the next chapter, we detail our experimental setup.

Chapter 5

Experimental Setup

In this chapter we discuss the setup of the various experiments. In Section 5.1, we detail what attributes we need to showcase in the cooperative case, and we provide a set of environments that capture these desired features. Then in Section 5.2, we discuss what properties for each of the environments we would need in a competitive setting, and we provide environments which capture these. We examine these two cases separately to evaluate the potential benefits and limitations of our approach, as reliability may be beneficial when working with a teammate. However, this reliability in behaviour can become a liability, as it makes an agent more predictable. An opposing agent may find it easier to plan against actions that follow a consistent pattern, potentially turning an advantage into a disadvantage. Finally, in Section 5.5, we detail the underlying hyperparameters for each of our various approaches.

5.1 Cooperative Case

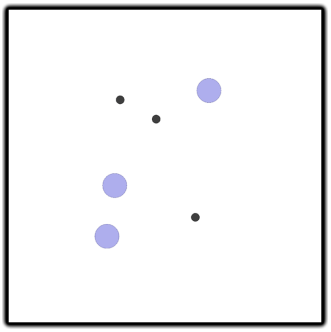
The cooperative setting is our primary concern. This setting enables us to determine whether our proposed entropy-averse agents are more adept at collaborating than entropy-embracing agents in completing the given tasks. If agents are able to overcome the non-stationarity, they should have a better chance at performing well and constructing policies that implicitly take the other into account.

5.1.1 Environments

To demonstrate the difference between entropy-embracing agents and entropy-shunning agents, we need three components.

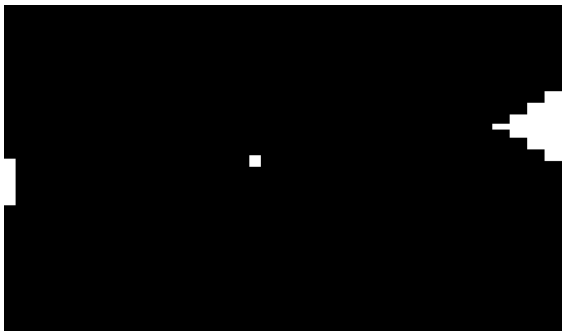
Coordination Firstly, we need a benchmark environment that requires navigation in a coordinated manner between agents. In real world tasks agents need to be able to navigate environments, whilst avoiding collision with other moving agents. To this end, we first consider the widely-used environment of Simple Spread [Lowe *et al.* 2017; Yu *et al.* 2021]. Here, agents must reach a set of two landmarks, but they are penalised if they collide. The agents are rewarded based on how far the closest agent is to each

landmark (i.e. the agent with the minimum sum of its distance to each landmark). Agents can move in the four cardinal directions or take no action (stay still). An agent's observation consists of their velocity, their position, relative distance to the landmarks and the relative position of the other agent. A visual for this environment is given in Figure 5.1a.



(a) A frame of the start of a game of Simple Spread. In this game the number of agents can vary (here there are three) and they are represented by the lilac circles. The landmarks that the agents must reach are represented by the black dots. We have placed a border around the game for clearer visual reasons.

(b) A frame to illustrate the starting setup of a game of Space Invaders. Here walls, which block shots, are represented in orange. These walls break as bullets hit them. Above the walls are the enemy agents which occasionally drop bombs as they move downwards in a zigzag fashion. At the bottom are the two cooperating agents (yellow and green) which must kill the enemies.



(c) This is a visual of the start of a game of Cooperative Pong. Here the two heterogeneous agents are on the left and right-hand side of the screen. The ball that they must continue to bounce between them is the square in the middle.

Figure 5.1: Cooperative Environments

Moving Agent Problem Secondly, we require an environment that demonstrates agent reliability and exhibits the "moving agent" problem to a greater extent. We recall that the moving agent problem mentioned in Section 3.4 refers to when one agent conditions its policy on another agent which is still learning and changing its policy. Thus,

we use Space Invaders since if one of the agents dies, both of the agents lose a life, and the game resets. The agents' objective is to eliminate all the enemies that zigzag down from the top of the board. The enemies are organised in multiple rows and drop bombs that can kill either of the agents. The game ends if the agents lose all three of their shared lives or if the enemies reach the bottom of the board. To ensure consistency with Mnih *et al.* [2016], we also clip rewards to the range of $[-1, 1]$. There are five actions available to the agents in this game: stay still, fire, move up, move right, move left, and move down. These actions are defined by the PettingZoo library. Agents are required to learn when they are responsible for prematurely terminating the game, avoid it, and at the same time, be a useful teammate by learning their own policy to help achieve more rewards. Furthermore, both agents rely on each other not to end the game prematurely for them to perform well. We provide an illustration of this game in Figure 5.1b.

Heterogeneous Control Finally, we need an environment where not all agents are the same and requires heterogeneous agents to coordinate to perform well. To do this, we use Cooperative Pong as the agents are heterogeneous, with one agent similar to the Atari Pong agent and the other a triangle. The goal of this game is to keep the ball going for as long as possible, which corresponds to the maximum number of steps achievable. The game concludes when the ball goes out of bounds on either the left or right edge of the screen. At the start of each environment reset, the ball may begin moving either left or right. For both agents, they have the option to move up or down. We give a reference frame of this game in Figure 5.1c.

5.1.2 Environment Settings and Observation Modifications

We demonstrate the performance of our agents in these environments, running our agents over twenty different seeds and reporting the performance using 95% confidence intervals. The specific implementations of each of these environments come from PettingZoo[1]. PettingZoo is a library for MARL environments [Terry *et al.* 2021], and we used the environment wrapper SuperSuit [Terry *et al.* 2020a][2]. For all the settings, we only use two agents. We used frame stacking[3], and stacked four frames in all domains with all environments vectorised. The environments and their maximum number of steps are listed below, with any specific changes noted.

1. Simple Spread: In this environment, we employ two agents and utilise version 2. The observation space comprises a vector that includes the agent's position, velocity, the other agent's relative position and velocity, and the relative position of landmarks. The discrete action space allows the agent to move in one of the

[1]https://pettingzoo.farama.org/

[2]https://github.com/Farama-Foundation/SuperSuit

[3]Frame stacking is when observations are concatenated, such that the last n number of observations are in the input. Framestacking effectively forms a moving window of the game, and hence, can be considered somewhat a form of "memory" and motion.

four cardinal directions or remain stationary. This experiment runs for 25 million steps, and we apply the same reward clipping used in Space Invaders.

2. Space Invaders: Space Invaders: In this setting, we utilise observations based on the Atari game's simulated RAM, which is structured as 1024 bits in the virtual memory of the Atari console. This RAM observation provides insights into the internal states and dynamics of the game environment beyond mere visual frames. We adopted a frame skip strategy of four, mirroring the technique in Mnih *et al.* [2015], and implemented reward clipping within the range $[-1, 1]$, using version 2. The experiment was carried out over 40 million steps. To aid in the normalisation process, we scaled the RAM observation values by dividing them by 255, since the values range from 0 to 255.

3. Cooperative Pong: We resize the observations to 84×84 pixels using only the blue channel. This experiment ran for 32 million steps. We divide the pixel values by 255 to normalise observations.

In Simple Spread and Cooperative Pong agents receive a team reward as this is the environment's reward. However, for Space Invaders, each agent receives a reward based on their own performance – which can allow for agents to act slightly greedier to the detriment of the teammate. For example, they can come out of cover and continuously shoot until the game ends, which will result in the agent losing but achieving a relatively high reward. Although, this is not the maximum reward that they can achieve if they were to act considering a longer time horizon.

5.2 Competitive Case

To determine whether reliability is a beneficial component in a competitive setting we compare our entropy-shunning agents and entropy-embracing agents in two competitive environments. We conduct this analysis to investigate whether being reliable is a disadvantageous trait, as it may make an agent more predictable and vulnerable to being countered by another agent. An agent with a more exploratory behaviour, on the other hand, might be more open to taking a variety of actions and would be less easily modeled by the opposing agent, as this less exploratory agent would be more easily modelled or its actions in a given state anticipated.

Remaining Flexible It may potentially be possible for agents to perform better if they are flexible in their action choices by having a flatter probability distribution over their actions. This flatter distribution would allow an agent to explore more, and potentially be harder to predict and plan around. This continuation in exploration may allow the entropy-embracing agent to take advantage of a more decisive agent. Alternatively, it may be possible that, as one agent finds an optimal strategy that exploring further and deviating from this strategy is strictly suboptimal. The next section empirically demonstrates this in the boxing domain. In this game, the agents are in a boxing ring with the objective of scoring higher than their opponent. Points are awarded for landing punches, with a long jab worth one point and a close punch worth two points.

The reward of one agent is the penalty of the other. Lastly, each agent has access to a diverse set of eighteen different actions. A visual representation of Boxing is given in Figure 5.2a.

The rationale behind selecting this environment to demonstrate the potential advantages of being more deterministic over taking a variety of actions is deeply rooted in the essence of multi-agent interactions. With the increasing complexity of MARL tasks, entropy regularisation plays a pivotal role in defining an agent's behaviour. Being deterministic or having a higher inclination towards specific actions signifies the agent's certainty in its policy. On the other hand, maintaining a diverse action set represents a high level of uncertainty. The question is, which approach leads to better cooperative or competitive outcomes? Here are the considerations we made:

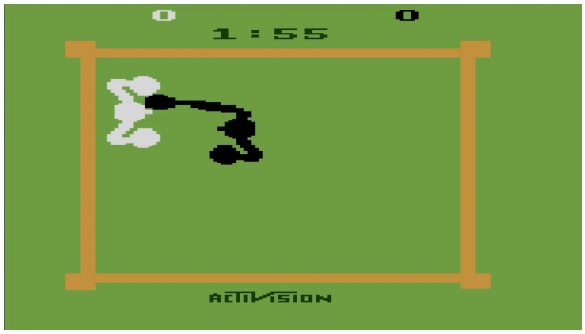
- If their opponent backs into a corner then you may want to press the advantage as they are less mobile now, rather than taking an exploratory action, and allowing them to get into a better position.
- If an agent notices a consistent suboptimal movement they may avoid it, for example, being too close to their opponent and instead remaining mobile and attacking when they come towards them.
- Being flexible and taking various actions could be beneficial as it makes the agent a less reliable enemy therefore, it is more challenging for another agent to play against them.

Entropy regularisation in MARL can fine-tune the balance between exploration and exploitation. While exploration helps in discovering new strategies, there comes a point in many environments where the agent must solidify its policy to reap maximum rewards. The aforementioned points emphasise situations where a tempered approach to entropy can benefit an agent in MARL scenarios.

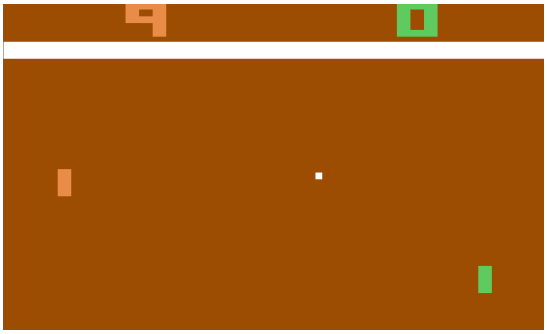
Contrasting RL Environments The other environment we consider is Pong, which is illustrated in Figure 5.2b. We choose Pong because it is a widely-used Atari game that has been used when evaluating RL algorithms. A secondary motivation is that this environment represents the competitive version of Cooperative Pong. In Pong, the objective is to score points by hitting a ball past the opponent's paddle. The player scores one point if the opponent misses the ball, and the first player to reach 20 points wins the game. The paddle can move up and down to hit the ball back to the opponent. However, agents have access to six different actions. A frame from the start of this game is given in Figure 5.2b

5.2.1 Environment Settings and Observation Modifications

In the competitive setting we also make use of four frame stacking, we used the RAM observations, and we divided the observation values by 255 to normalise them. Lastly, the Boxing domain had a total number of 60 million steps, whilst the Pong setting had 40 million steps.



(a) A visual from the start of the Atari Boxing game. Here the two agents are inside a boxing ring. The two agents are represented using black and white, and their respective scores represented at the top in their corresponding colour.



(b) This is a frame from the Atari game Pong. Here the two opposing agents are on the left and right-hand side of the screen, and their respective scores represented at the top in their corresponding colour. The agents compete against each other using the ball to score which is the square pixel in the middle.

Figure 5.2: Competitive Environments

5.2.2 Round-Robin Tournament

To evaluate the performance of each agent, we allow different approaches to compete against one another. This tournament format, in which every approach faces all other approaches, is known as a Round-Robin tournament. We conduct this to assess the performance of each approach relative to the others. Specifically, we examine the performance of an entropy-embracing agent against an entropy-shunning agent. Moreover, we permit each agent to play in every position to circumvent position bias. Position bias refers to the potential challenge an agent may face in learning as either the first or second player. Hence, during each stage of the Round-Robin tournament, every agent plays as both player one and player two. A more comprehensive explanation of this experiment is provided in Section 7.1.3.

5.3 Metrics

Before we outline the various metrics we use to analyse the various methods, we introduce a specific method we use in performing some of our calculations. We make use of an Exponentially Weighted Moving Average (EWMA). This method is commonly used to help filter out the noise and assists in noise reduction. Given that MARL, by nature, has a high degree of non-stationarity, we use this to prevent outliers from significantly influencing our results. Furthermore, we use this method because it can effectively reveal overall trends in the inherently non-stationary MARL experiments, while capturing the dynamics by placing greater emphasis on recent data points. This approach helps filter out noise and prevents outliers from significantly influencing our results, making it well-suited for analysing fully-decentralised MARL data. The general formula for this approach can be represented as such:

$$y_t = \frac{\sum_{k=0}^{t} (1-\alpha)^k r_{t-k}}{\sum_{k=0}^{t} (1-\alpha)^k}, \tag{5.1}$$

where r_k is the reward at time step k, α represents the smoothing factor, and y_t represents the smoothed reward. In Equation 5.1, we demonstrate this with rewards. However, this can also be done for the number of steps in an episode. Before calculating the EWMA for a set of values, we take the mean of the agent's rewards and lengths for each episode. For example, we construct the mean reward for an agent's tenth episode by taking the average reward that the agent received at the tenth episode for each of the seeds that the agent was trained on. Using this averaged reward, we calculate the exponentially weighted moving average. In our particular case, we do this using $\alpha = 7.997 \times 10^{-5}$, as this demonstrates the performance trend most clearly. Furthermore, using this value for α allows us to dampen the noise that outliers may have when we demonstrate an agent's performance. We also use the EWMA to calculate the standard deviation. This EWMA allows us to accurately demonstrate how much variation exists in agent performance and allows us to smooth out the standard deviation to show the general trends effectively. We use the standard deviation and our mean to calculate each approach's 95% confidence intervals. These confidence intervals allow us to provide statistically significant evidence of our various approaches and their performance.

The specific metrics we capture are as follows:

1. **Agent learning curves:** These are overall agent performance in specific environments. These curves use a smoothing term of $\alpha = 7.997 \times 10^{-5}$. They demonstrate an agent's mean reward for that episode across all seeds. The error bars for these curves are the 95% confidence intervals. These curves allow us to readily assess an agent's learning, their end performance and sample efficiency. We can see the sample efficiency as the agent learns based on the steepness of the curve and how quickly (in terms of episodes) it reaches a certain threshold of reward. We measure this performance (and reward) using the base agent as our comparison

2. **Mean episode performance:** These plots have an agent's overall mean performance throughout training and their respective 95% confidence intervals. We use a smoothing term of $\alpha = 7.997 \times 10^{-5}$ for these plots. These plots allow us to compare various agents or hyperparameters with one another easily. An explanation for why we do not use the area under the curve as a measure is provided in the Appendix. The primary reason is that the episode lengths are not uniform across all agents. Although the agents are trained for the same number of steps, two things can occur. First, the agent may have hyperparameters that hinder learning, causing the agent to stop training early when the network fails to continue learning. This issue generally arises with parameters that lead to *NaN* (not a number) values. These *NaNs* typically occur due to exploding gradients. This phenomenon happens when the probabilities of actions become increasingly smaller, approaching zero, leading to issues with numerical stability. Second, even if the parameters do not cause problems with *NaNs* but are still suboptimal, the agent may experience many episodes of shorter length (fewer steps per episode) com-

pared to a better-performing agent. This effect of many short episodes results in an agent having a similar cumulative reward as its curve has many more x-values (episodes) associated with it despite the fact that the y-values (the reward) are much lower. Therefore, using the mean reward helps to avoid this issue when demonstrating agent performance, as agents that exhibit this problematic trend perform significantly worse.

3. **Time to reach 90%:** To demonstrate an agent's sample efficiency compared to the base approach, we compare each agent's number of episodes required before reaching the 90% percentile of the base agent's rewards. This graph allows us to see how various methods compare their sample efficiency easily. For these graphs, we use a lower smoothing factor of $\alpha = 1.998 \times 10^{-3}$. This value allows us to weigh shorter periods more effectively demonstrating the agent's upper performance bound better.

4. **The difference in mean reward:** To represent the difference in the agent's mean rewards for the various approaches, we do the following. We calculate the base approach's mean reward over all seeds, and this reward represents the mean reward per episode for the agent. Here, the number of seeds vary in the cooperative and competitive case, where we use 20 and 10 seeds for the cooperative and competitive settings, respectively. However, we subtract the base approach's mean from the other approaches and divide it by the base reward. This format allows us to demonstrate the difference in the mean reward for all approaches, compared to the base approach, as a percentage. Equation 5.2 provides a concise demonstration of how we calculate this change:

$$\text{agent improvement} = \frac{\mu - \mu_{base}}{\mu_{base}} \times 100, \tag{5.2}$$

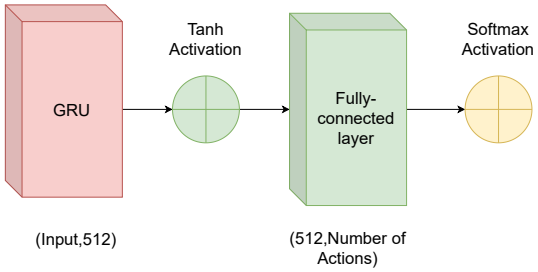
   where, μ represents the mean agent reward over all seeds and μ_{base} is the mean base approach reward over all seeds.

5. **The distance between agents:** We follow the same steps for agent learning curves in this case. However, we subtract agent two's rewards from agent one's. This approach allows us to see the difference between agent performance. If the reward is less than zero, agent two has a larger reward; if it is greater than zero, then agent one has a larger reward. We use this metric primarily in the competitive setting to demonstrate how fair games are.

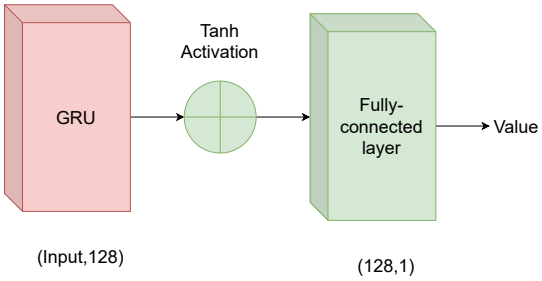
5.4 Network Architecture

Base Network For all experiments, we used a separate network for the policy and value function. The motivation for this comes from Andrychowicz *et al.* [2020], who performed a large-scale empirical study and found that having separate networks is beneficial. The choice and structure of our architecture is primarily based on prior work [Mnih *et al.* 2015; Schulman *et al.* 2017b; Huang *et al.* 2022]. The inputs stayed the same for both of these networks, except in the case of Cooperative Pong, which does not have a RAM observation. The environment determined our input sizes for

the policy and value function for the agents. However, excluding this, the network architectures are the same. For the policy network (visualised in Figure 5.3a), we had a GRU layer that took in the observation and had an output of 512 nodes. This GRU connects to a fully-connected layer with an input size of 512 and the output size is the number of actions that can be taken in the environment. We chose to use a memory layer because it has been used in prior work in multi-agent reinforcement learning, such as that of Foerster *et al.* [2018]. The value network (visualised in Figure 5.3b) is similar to the policy network. However, it has 128 nodes instead of 512 and outputs a single value rather than a vector containing action probabilities. The activation between the GRU layer and the fully-connected layer are *tanh* functions for both networks. The choice of *tanh* as our activation is based on prior work which recommends it and states that it may be better suited when performing entropy regularisation [Andrychowicz *et al.* 2020]. This setup forms our base network.



(a) An illustration of the pipeline that we use for our policy network.



(b) A visual representation of the pipeline that we use for our value network.

Figure 5.3: The base neural network architecture for our all of our experiments.

Cooperative Pong Modifications In the case of Cooperative Pong, the general network is very similar. We follow the work of Mnih *et al.* [2015 2016] when constructing this modified layer to handle the visual input. Here, we add a processing component at the start for both the policy and value functions. We describe the initial preprocessing that is performed by this additional component of the network with an illustration of it given in Figure 5.4. After every convolution, we have a *ReLU* layer as our activation function. This extra layer consists of a convolution layer with 4 input channels, 32 output channels, a kernel size of 8 and a stride of 4. After which, we have a second convolution layer with 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2. Following this, the final convolution layer has 64 input channels, 64 output channels, a kernel of size 3 and a stride of 1. This output is flattened and passed through a fully-connected layer with 3136 input nodes and 512 output nodes. This output then has a ReLU activation performed on it and is then passed through the same network that we used for all other environments (our base network).

5.5 Reproducibility

In the interest of reproducibility, we list all hyperparameters we used in Table 5.1 for the base PPO algorithm, and we detail other design choices made when performing our

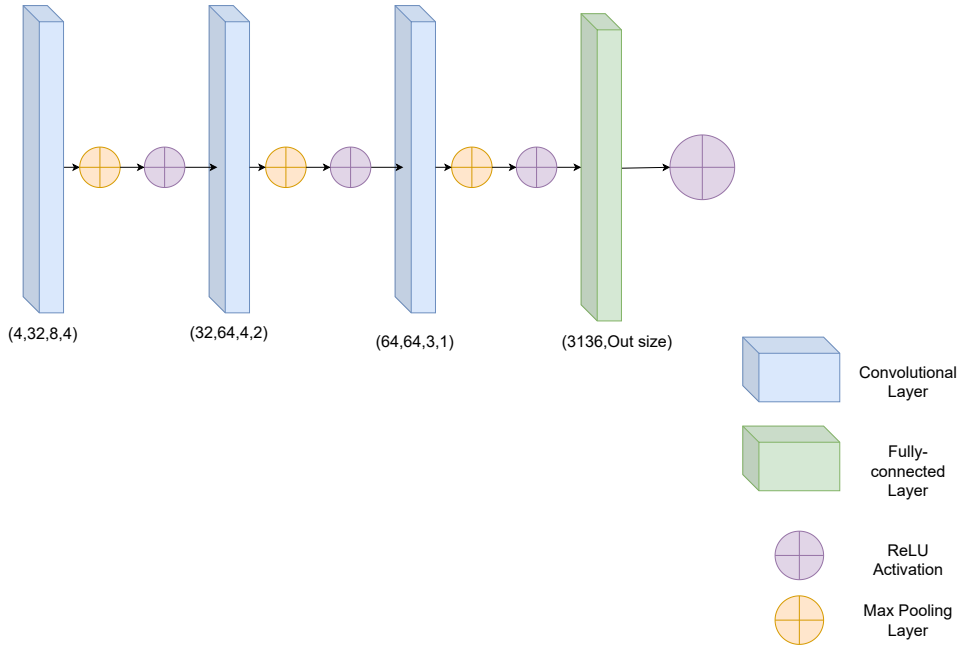


Figure 5.4: An illustration of the additional preprocessing network used for the Cooperative Pong environment. In this visualisation the tuple under each convolutional layer represents the number of input channels, the number of output channels, the kernel size and the size of the stride, respectively.

experiments. We use a base *PPO* algorithm modified to perform in a fully-decentralised fashion. The repository we use to create our model was CleanRL [Huang *et al.* 2022][4]. There are several libraries available that implement PPO, such as Stable Baselines [Hill *et al.* 2018], Spinning Up[5], and Ray RLlib[6]. However, we opted to use ClearnRL because of its simple single-file implementation. It provides a simple implementation of PPO, which allows us to easily implement our necessary augmentations for our baselines and alternative methods. We compare our approaches against base *PPO*, standard Shannon entropy and *ADE*. We perform an equal amount of hyperparameter tuning for each of these approaches' variables and use the best hyperparameters for each. The parameters found in Table 5.1 are common for all approaches that we examine, and they are used for each of our domains.

In PettingZoo, the number of environments effectively doubles when using a vectorised environment. This doubling arises because each agent for an environment not only receives its own observation from that environment but also an additional indicator variable specifying to which agent a particular observation pertains. Thus, in practical terms, each agent trains using 16 environments, given that all our experiments involve two agents. We use the indices for each agent to collect their respective observations

[4]https://github.com/vwxyzjn/cleanrl
[5]https://github.com/openai/spinningup
[6]https://docs.ray.io/en/latest/rllib/index.html

Hyperparameter	Value
Total Timesteps	Environment Specific
Learning Rate	2.5×10^{-4}
Number of Environments	32
Number of Steps in Each Environment	256
Anneal Learning Rate	True
General Advantage Estimate (GAE)	True
γ	0.99
GAE_λ	0.95
Number of Minibatches	4
Update Epochs	4
Advantage Normalisation	True
Surrogate Clipping Coefficient	0.1
Clipped Loss for Value Function	True
Entropy Coefficient	0.01
Value Function Coefficient	0.5
Maximum Norm for Gradient Clipping	0.5
Batch Size	$\frac{num-envs}{2}$ * num-steps

Table 5.1: Table containing all PPO hyperparameters and their values that we used when conducting all of our experiments.

from the vector of observations. This collection of indices is necessary as, at each step, agents receive a set of observations and which observations belong to each agent is a vital component of the observability in MARL. When agents interact with the environment we place their actions into a vector based on their respective indices from the observations. Finally, we found that using a separate network for the policy and the value function performed better.

5.6 Hyperparameters

When conducting hyperparameter tuning to compare the various approaches, we follow these steps. For each approach, we test a range of eight potential hyperparameters based on our modifications to entropy regularisation. We run each approach with every set of hyperparameters using twenty different seeds to demonstrate its overall performance. Table 5.2 displays the values that we test. If a specific approach has only one value in the table, it represents the ξ value, which is the coefficient for our entropy regularisation term. If there is a second value (as in the case of RECO), it represents the κ value, which is the absolute value used to clip our entropy regularisation term. Additionally, we indicate a label which is either A or B. This label is for approaches which use a clipping term and it is referenced when showcasing our results in Chapters 6 and 7. Here we search over two values for κ as the size of the variable after being clipped and multiplied by ξ is quite small. Furthermore, larger κ values lead to larger

gradients when the entropy of some actions is near zero and hence clipping on larger values would lead to greater instability.

Approach	1	2	3	4	5	6	7	8
ADE	-0.001	-0.005	-0.01	-0.05	-0.1	-0.15	-0.2	-0.25
Shannon	-0.001	-0.005	-0.01	-0.05	-0.1	-0.15	-0.2	-0.25
RECO	$0.001, 20$	$0.001, 2$	$0.01, 20$	$0.01, 2$	$0.05, 20$	$0.05, 2$	$0.1, 20$	$0.1, 2$
ADE(neg)	$0.001, 20$	$0.001, 2$	$0.01, 20$	$0.01, 2$	$0.05, 20$	$0.05, 2$	$0.1, 20$	$0.1, 2$
RECO(neg)	$-0.001, 20$	$-0.001, 2$	$-0.01, 20$	$-0.01, 2$	$-0.05, 20$	$-0.05, 2$	$-0.1, 20$	$-0.1, 2$

Table 5.2: This table displays the hyperparameters that we searched over for each approach. For approaches with a single value, the displayed value represents the ξ value, which is the coefficient for exploration. For approaches that require clipping, the displayed value consists of two parts separated by a comma: the first part represents the coefficient for entropy, and the second part represents the κ value. To assist in graphing the performance of various approaches, we represent 20 using A and 2 using B.

5.7 Summary

In this chapter, we describe the experimental setup used to validate and compare our approach with the standard method of promoting entropy. We have presented five distinct environments encompassing cooperative and competitive settings, detailing each environment's rationale, modifications, and relevance. Furthermore, we have outlined the modified algorithm employed by our approaches and provided essential metrics to evaluate the performance and learning of the agents. Although we have utilised a variety of metrics, environments, and hyperparameters, there may still be potential edge cases that need to be explored. Nonetheless, we have comprehensively demonstrated and contrasted our approach with the current standard. The following two chapters will analyse our method's performance in cooperative and competitive scenarios.

Chapter 6

Cooperative Results and Discussion

In this chapter, we present results for the cooperative domains. We demonstrate the performance of each approach (entropy-embracing, entropy-shunning and the base approach) in Section 6.1. In this section, which includes an analysis of the algorithms in terms of their sample efficiency and rewards that they accumulate. Following this, in Section 6.2, we illustrate the effects of entropy on agent performance, and we do this by examining entropy-embracing agents. We demonstrate the sensitivity and need for careful parameter tuning for entropy-shunning agents in Section 6.3. Lastly, for all figures, we use the same labelling for each set of hyperparameters as set out in Table 5.2. Hence, when clipping we use let $A = 20$ and $B = 2$. Lastly, for all cooperative domains, we use twenty seeds when running all of our experiments.

6.1 Rewards and Sample Efficiency

6.1.1 Simple Spread

First, we demonstrate how the various approaches performed with the best hyperparameters we found for each as indicated in Table 5.2. Figures 6.1 illustrate the reward the agents received for each episode during the course of training. These figures showcase the mean reward over the episodes and the 95% confidence interval of the reward. Moreover, in Simple Spread, both agents receive the same reward, making their rewards identical.

In Figure 6.1, it is evident that most of the approaches achieve a similar performance with only slight variations. Specifically, RECO and ADE(neg) perform almost identically to the base approach. However, in this simple domain, we observe that the standard approach of adding entropy (represented by ADE and Shannon entropy) performs slightly worse than both the base approach and ADE(neg). Additionally, RECO(neg) exhibits greater variance compared to the other approaches, which may cause slight instability in performance. Figure 6.2 illustrates the agent's performance over the course of the last 1000 episodes. In this figure, we observe that the approaches generally exhibit similar performance, with ADE, ADE(neg) and RECO achieving a final mean similar to that of the base approach. Therefore, in this case, the type of entropy used may have

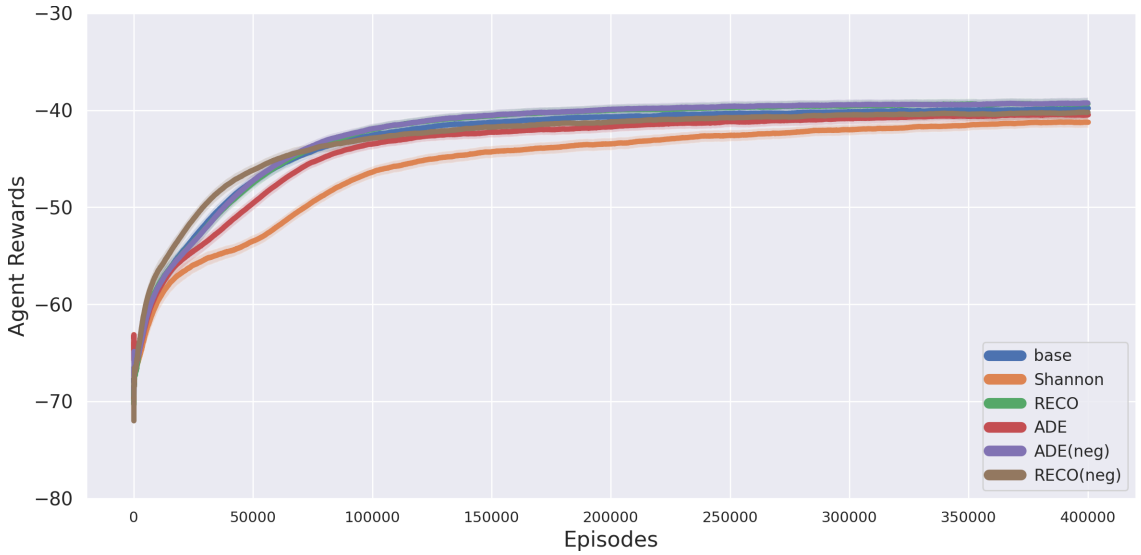


Figure 6.1: Rewards for each of the various approaches in the Simple Spread environment (higher is better). This graph is truncated to help visualise the learning process of agents early on as agents converge to the final values at the end of training shown in this figure.

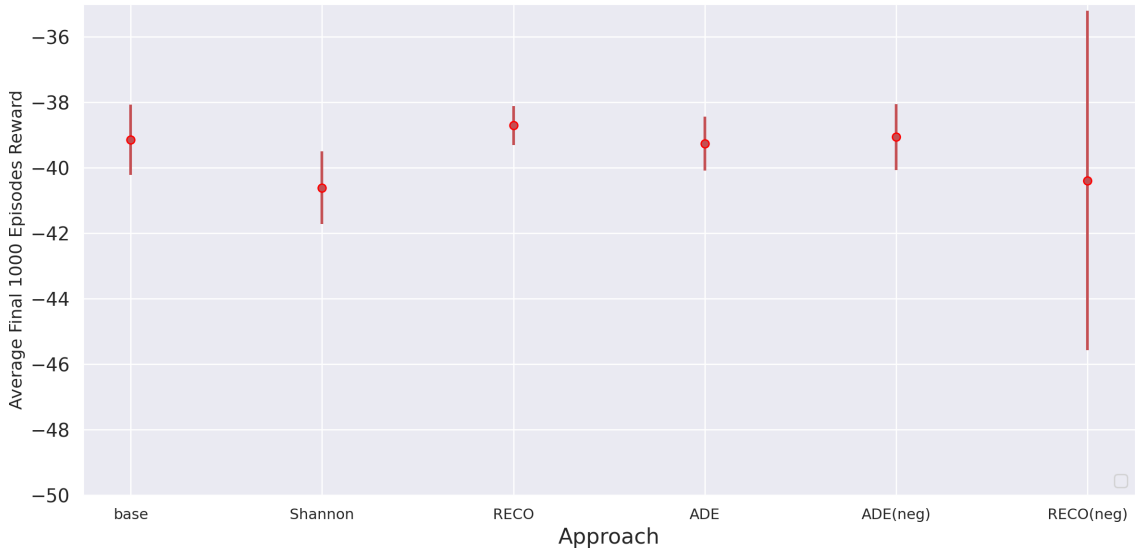


Figure 6.2: Mean reward and confidence interval for the last 1000 episodes for method in Simple Spread (higher means are better). We calculate the means using the final 1000 episodes during the agents' training to demonstrate their final performance at the end of training. This allows us to assess how well the agents have learned to perform the task after being exposed to the environment for an extended period of time.

varying effects on the final performance and could be considered a separate hyperparameter in certain cases. Furthermore, Figure 6.1 shows that RECO, ADE(neg), and the base approach exhibit similar sample efficiency, as demonstrated by their learning curves at the start of training. However, Shannon entropy, ADE, and RECO(neg) exhibit slightly worse sample efficiency, with Shannon entropy performing the worst. In

this simple environment, improved sample efficiency is desirable as it enables agents to train for fewer episodes while achieving the same performance. Therefore, although the varying approaches have similar overall performance in this domain, the agents with better sample efficiency are preferable.

6.1.2 Space Invaders

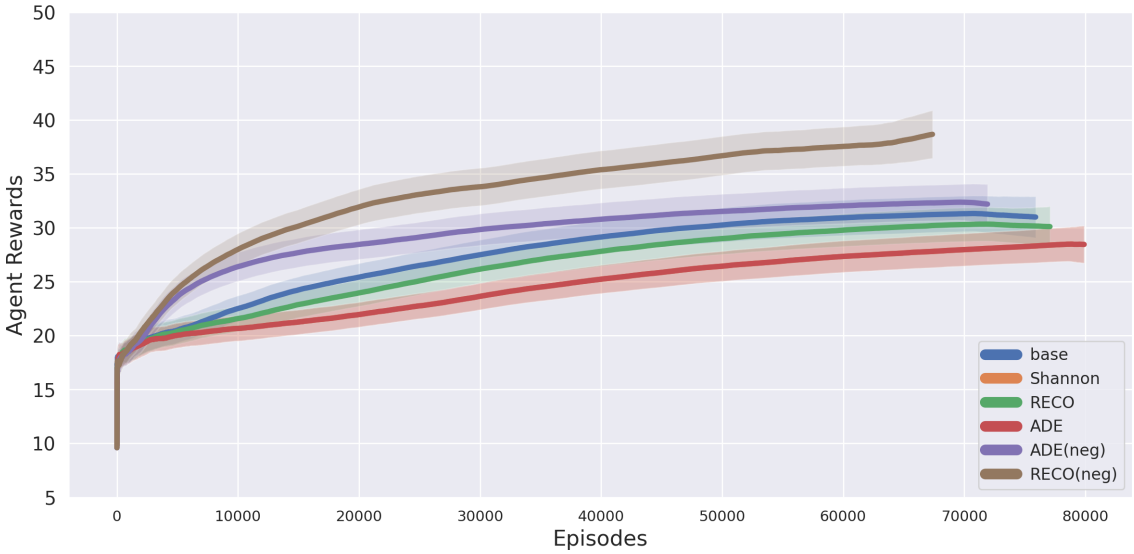


Figure 6.3: Rewards for each of the various approaches in the Space Invaders environment (higher is better). Furthermore, we note that Shannon Entropy and ADE perform identically in this case. Agents can have a different number of steps in each episode (as outlined in Section 5.1) hence, although they train for the same length of time they have experienced the same number of steps.

Figure 6.3 clearly demonstrates that the entropy-embracing agents perform worse overall than the base approach, with ADE and Shannon entropy exhibiting identical performances. These identical performances can be attributed to a key aspect of the ADE method as stated by Zhou *et al.* [2020], which is a modified form of Shannon entropy designed to prevent overconfident agents from prematurely converging and thus maintain exploration. However, in this environment, the non-stationarity and the ringing agent problem (mentioned in Section 2.4) are likely to be exacerbated, making it difficult for agents to gain confidence in their responsibilities. The constant exploration by the participating agent makes it challenging to settle on a policy. Consequently, the ADE method becomes similar to Shannon entropy, as the agent never truly gains confidence in their actions due to the inherent non-stationarity and complexity of the fully-decentralised Space Invaders environment using PPO as the algorithm. On the other hand, the number of episodes required for the agents to improve their performance demonstrates that entropy-shunning agents achieve significantly better sample efficiency than their entropy-embracing counterparts. Specifically, RECO(neg) and ADE(neg) exhibit much faster learning rates than the base approach, as illustrated in Figure 6.4, which shows the fraction of episodes required by an approach to reach the 90th percentile of performance out of the total number of episodes spent learning. In

this regard, the two entropy-shunning agents achieve this milestone much earlier, with RECO(neg) requiring less than 20% of the episodes used in training. However, even at the end of training, entropy-embracing agents fail to reach the mean rewards achieved by the base approach.

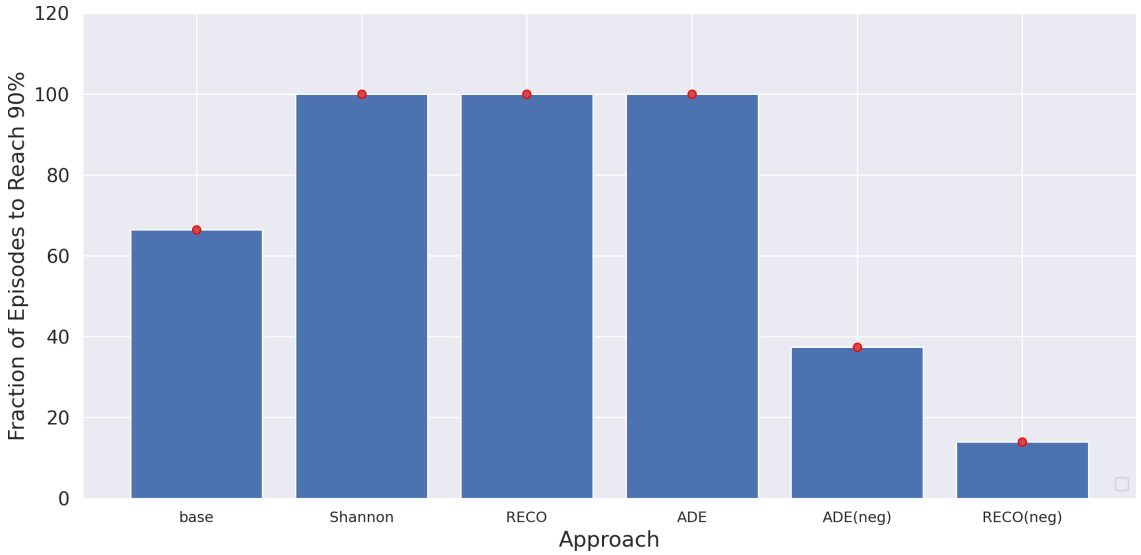


Figure 6.4: Fraction of episodes required to reach the 90^{th} percentile of the base's performance in Space Invaders, represented as a percentage of the total number of training episodes (lower values are better). Here, approaches which reach the 100^{th} percentile never reach the same performance of the base algorithm.

We provide a detailed illustration of the sample efficiency for each approach in Figure 6.4. This figure demonstrates the percentage of episodes required before the agent achieved the 90% percentile of the base approach's performance. We can see clearly using this figure how much the improvement in sample efficiency is for any approach compared to the base approach. In this environment, ADE(neg) does slightly better than base but with improved sample efficiency as it requires almost half the number of episodes compared to the base approach. Furthermore, we see that Shannon, RECO and ADE are not able to reach th 90^{th} percentile, as their fraction of episodes required reaches 100^{th} percentile, which indicates that the agents never reached the base algorithm's 90^{th} percentile of performance. Lastly, we can see that the entropy-embracing agents suffer greatly in this task, as in both Figures 6.3 and 6.4. The reason they perform worse is, because as Figure 6.4 highlights, these agents never reach the base approach's 90% percentile whilst training. The base approach was unable to achieve a 90% value higher than approximately 31.68 during training. Figure 6.5 shows that the mean value of the other approaches during their final 1000 episodes of training, which lasted nearly 40 million episodes, was lower than this threshold. This explains why they did not surpass the base approach's performance. Furthermore, as Figure 6.3 demonstrates, these agents have lower means than the base approach, with the standard Shannon entropy and ADE approaches vastly underperforming this mean as their confidence intervals barely overlap. Hence, we can say entropy-shunning agents perform considerably better than their alternatives (entropy-embracing agents) in this en-

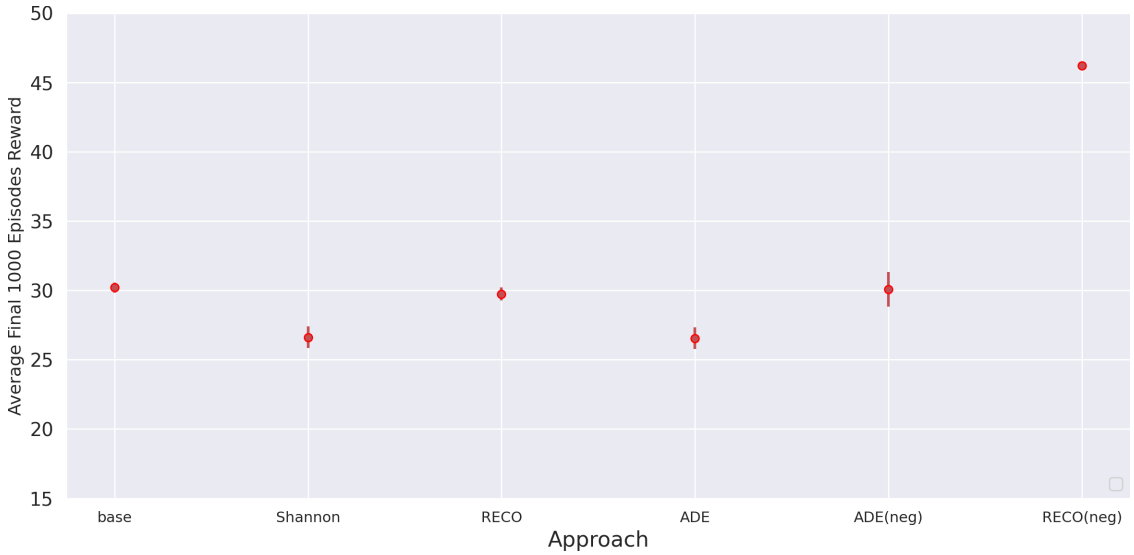


Figure 6.5: Mean reward and confidence interval for the final 1000 episodes for each approach in Space Invaders (higher means are better). We calculate the means using the final 1000 episodes during the agents' training to demonstrate their final performance at the end of training. This allows us to assess how well the agents have learned to perform the task after being exposed to the environment for an extended period of time.

vironment, given that they scored higher than them and demonstrated that they could outperform the base approach.

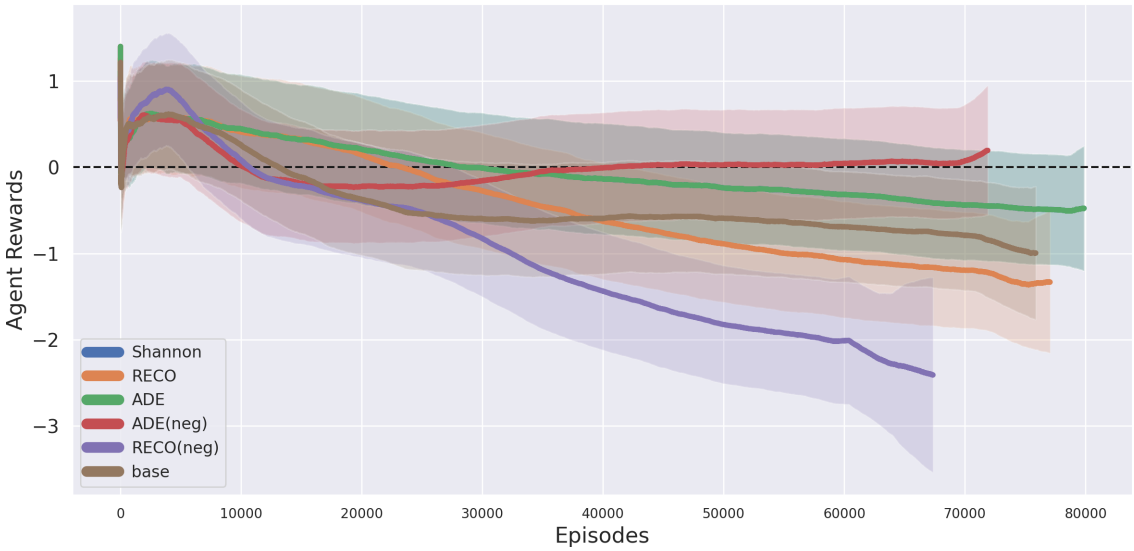


Figure 6.6: The difference in rewards between the first agent (indicated by positive values) and second agent (indicated by negative values), where negative values indicate that the second agent has more influence (values closer to zero indicate better balance). Agents can have a different number of steps in each episode (as outlined in Section 5.1) hence, although they train for the same length of time they have experienced the same number of steps.

However, the best-performing approach, RECO(neg), may potentially have one problem, namely the "lazy agent" problem. To determine which agent is a "lazy" agent, we calculate the difference between the two agents' rewards (shown in Figure 6.6). Should both agents have equal rewards, we can say that they are equal contributors in a cooperative setting, and the reward would be centred at zero. However, if one agent was responsible for all the reward, then we would see a reward curve that is strongly negative if the second agent contributed more or positive if the first agent contributed more. A negative value for distance serves as a useful indicator of agent performance and reveals which agent is more responsible for the overall performance. Rather than employing an absolute value, which would obscure the agent responsible for a larger share of the rewards, we retain the sign to signify this. Generally, we observe that the second agent (represented by negative values) tends to be more responsible for the rewards. Since we average the results of each experiment in this domain over 20 different seeds, this demonstrates a consistent trend in the agents' outcomes, particularly highlighting which of the two agents has a greater impact on the rewards.

Using Figure 6.6, we can determine whether one agent has more influence than another in this environment. Furthermore, this unequal contribution to reward is important to note, as both agents are identical in form and in function for this environment, therefore, agents can and ideally, should be equal contributors. Figure 6.6 demonstrates that in comparison to the other methods, RECO(neg) has the second agent achieving a notably greater reward than the first agent. This difference is essential as the other approaches typically have both agents contributing relatively similarly in the game as their differences are more centred around zero. Additionally, whilst other methods remain relatively centred around zero throughout training, we notice that the RECO(neg) agents have one agent contribute to an increasingly greater degree than the other. Therefore, whilst RECO(neg) performs the best in terms of reward and sample efficiency, this method may suffer from the "lazy" agent problem to a greater extent compared to the other approaches. Based on the better performance observed, it appears that the "moving" agent problem is less of a concern with the RECO(neg) method in this environment, which emphasises the need for agents to be reliable partners.

6.1.3 Cooperative Pong

We assess the performance of agents in the Cooperative Pong environment. This scenario, illustrated in Figure 6.7, shows that both entropy-embracing and entropy-shunning agents can achieve comparable results. In Cooperative Pong, an agent's primary focus is returning the ball to the other agent should it come towards them.

In terms of sample efficiency, Figure 6.8 shows that all approaches take a similarly long time to achieve the base approach's 90% percentile performance. However, RECO(neg) achieves it more rapidly.

With regards t agent performance, which is indicated in Figure 6.9, we see that all approaches have higher mean performance than the base approach, and hence, outperform it. We do note however, that Shannon entropy still has a large overlap in variance with the base approach.

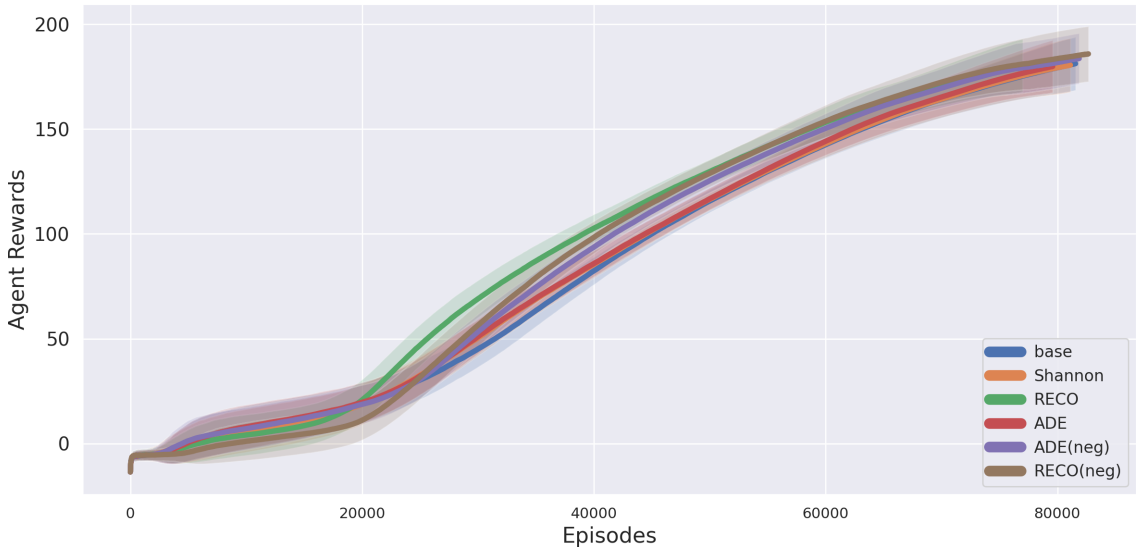


Figure 6.7: Rewards for each of the various approaches in the Cooperative Pong environment (higher is better). Here each agent receives the same reward at each step in the environment – similar to Simple Spread. Agents can have a different number of steps in each episode (as outlined in Section 5.1) hence, although they train for the same length of time they have experienced the same number of steps.

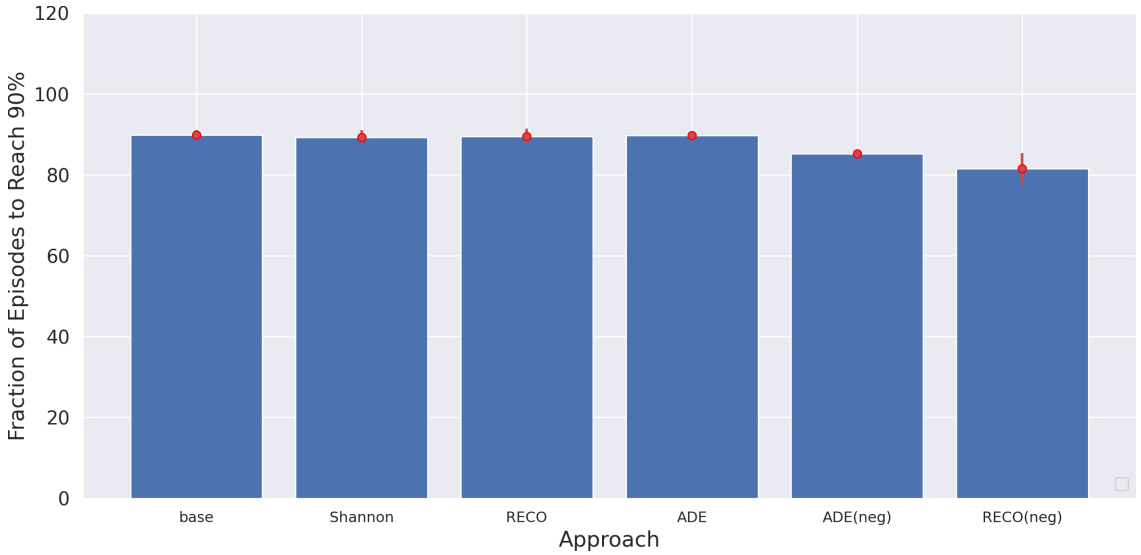


Figure 6.8: Fraction of episodes required to reach 90^{th} percentile of the base's performance in Cooperative Pong (lower is better). Here, approaches which reach the 100^{th} percentile never reach the same performance of the base algorithm.

However, given that their confidence intervals do overlap with the base approach for most of the other approaches, it is potentially possible the algorithms perform better in some cases, whilst worse in others.

Figure 6.7 shows that around thirty-thousand episodes, all approaches start to outpace the base approach, as evidenced by their higher rewards at this juncture. However, this quicker learning pace, as depicted in Figure 6.8, doesn't lead them to match the base

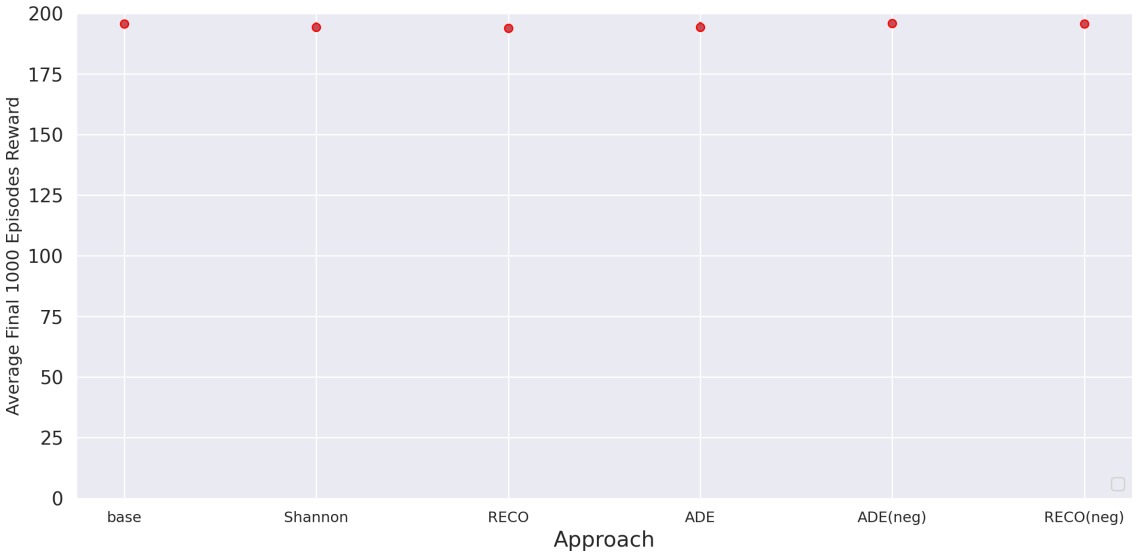
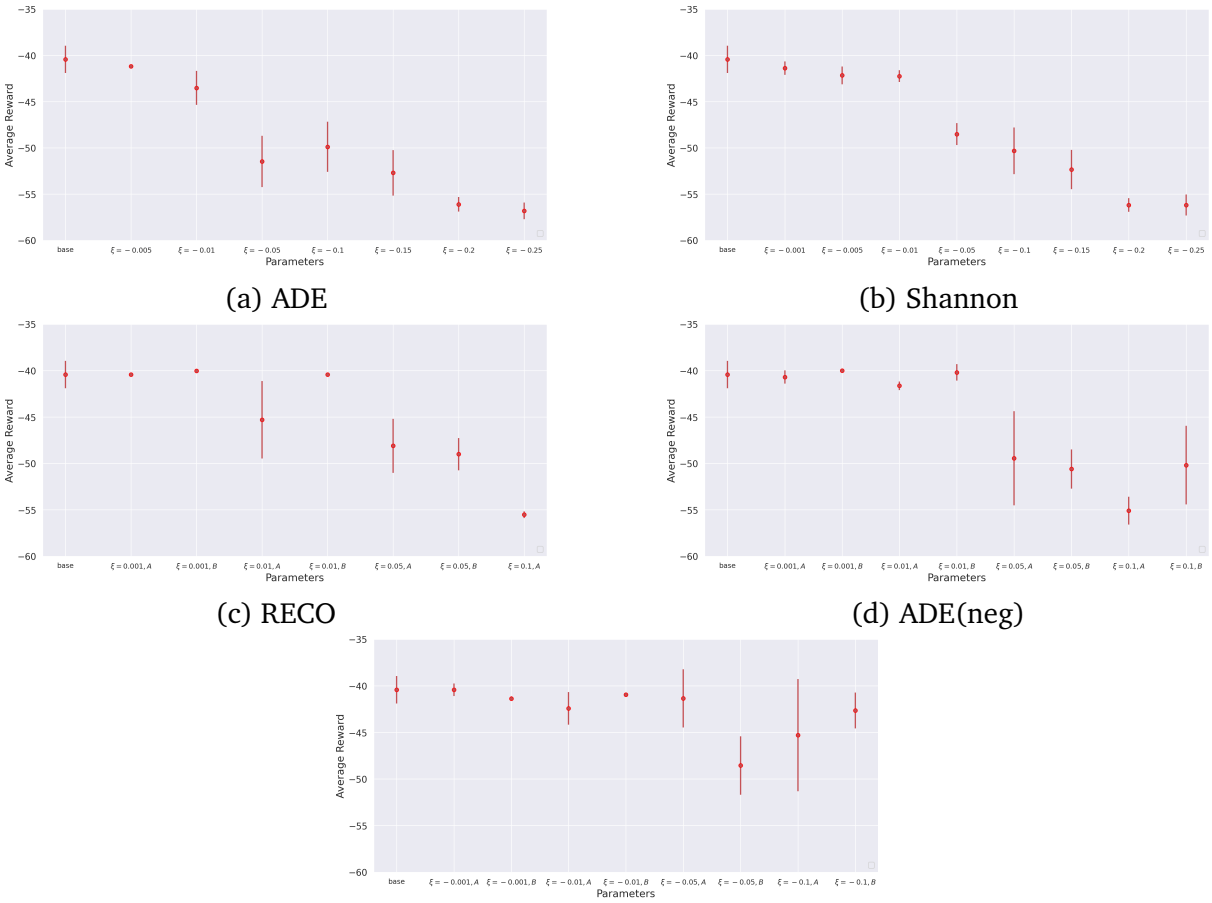


Figure 6.9: Mean reward and confidence interval for the final 1000 episodes for each approach in Cooperative Pong (higher means are better). We calculate the means using the final 1000 episodes during the agents' training to demonstrate their final performance at the end of training. This allows us to assess how well the agents have learned to perform the task after being exposed to the environment for an extended period of time.

approach's performance significantly earlier. By the sixty-thousand episode mark, all algorithms display comparable performance, as further illustrated in Figure 6.7. It's evident from these results that there are specific scenarios where both entropy-embracing and entropy-shunning agents can be beneficial. This observation is noteworthy since one of the primary motivations for incorporating entropy is its potential to enhance an algorithm's sample efficiency and end performance. Yet, when observing the similarities in final rewards across all approaches, it underscores a crucial point: in certain scenarios, entropy may not substantially influence agent performance.

6.2 Effects of Entropy

To better understand the effects of entropy in some environments, we examine the performance of the entropy-embracing agents using a subset of our key metrics from Section 5.3. We address how the impact of entropy can vary agent performance in a quantifiable manner. The parameter which determines the coefficient of entropy is ξ, as shown in Equations 4.4–4.8. However, for the RECO approach, given that our method gives much larger values, we clip the entropy value when the absolute value is greater than a specified threshold, and this is represented by the term κ, as seen in Equations 4.6–4.8. The larger values are a result of the function's design, with Figure 4.2 from Section 4.2.3 providing a reference for how much larger these values can be. We do this to illustrate why increasing entropy may not necessarily be desirable in certain situations.



(a) ADE

(b) Shannon

(c) RECO

(d) ADE(neg)

(e) RECO(neg)

Figure 6.10: Mean reward and confidence interval during the course of training (over 20 seeds) for various hyperparameters of the agents in Simple Spread (higher means are better)

Increasing Entropy In Figure 6.10, we see the effects as we increase the coefficient for the entropy bonus for each of the entropy-embracing agents in the Simple Spread environment. We see that increasing the entropy coefficient for ADE and Shannon entropy agents leads to decreasing performance. This relation can be seen most clearly in Figures 6.10a and 6.10b, where for each data point, as the coefficient of entropy increases, we see a corresponding decrease in agent performance for all but one data point at $\xi = -0.01$ for the case of Shannon entropy. In the case of ADE, we see a similar trend where in general, as entropy increases the performance of the agent subsequently declines. Notably, for the Simple Spread environment, the base approach which does not add additional entropy, other than that which the standard PPO algorithm uses, performs the best. This data point does better than $\xi = -0.005$ for ADE and $\xi = -0.001$ for Shannon entropy however, it still performs worse than the base.

Furthermore, we see with the RECO method that the same pattern exists. However, the additional clipping term may also strongly impact the agent's performance. This impact can be seen (Figure 6.10c) for $\xi = 0.01$, $\kappa = 20$, which performs better than if we clipped on $\kappa = 2$. A clear visual showcase of the effect of entropy on learning is

given in Figure 6.11, where we can see the bands during learning that indicate how, as entropy increases, the agent's performance and sample efficiency decrease. Therefore, in this simple environment, we can clearly see that increasing entropy is undesirable.

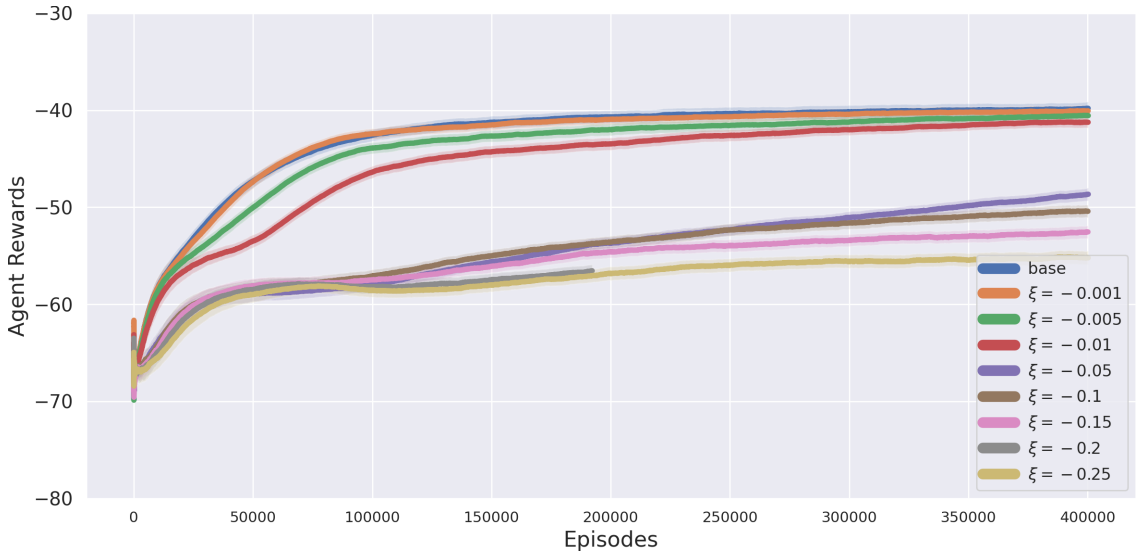


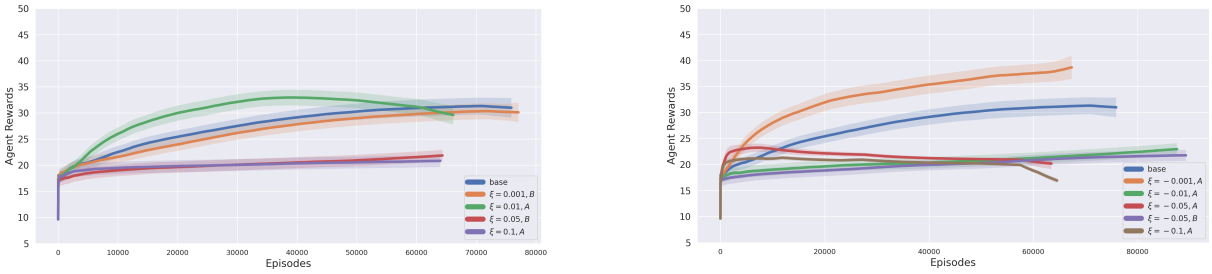
Figure 6.11: Rewards for each of the various hyperparameters of Shannon entropy in Simple Spread (higher is better). Here some seeds terminate earlier due to exploding gradients that may result from large values in the entropy regulariser term. This graph is truncated to help visualise the learning process of agents early on as agents converge to the final values at the end of training shown in this figure.

As a contrasting example to what is observed in Figure 6.11, we present Figures 6.10d and 6.10e. In Figure 6.10d, we notice that four different sets of parameters are able to achieve a similar mean performance as the base approach. Despite the differences in their parameters, such as the clipping value and the entropy regularisation coefficient, these configurations reach a similar final performance. Likewise, for the case of RECO(neg) in Figure 6.10e, we observe that three parameter combinations ($\xi = -0.001$, $\kappa = 2, 20$, and $\xi = -0.01$ $\kappa = 2$) attain comparable final performance to the base approach. Thus, in contrast to Shannon entropy and ADE, we find that as the coefficients become larger, the method's final performance does not deteriorate to the same extent. However, we do find that for some of these values, which are capable of performing as well as the base approach, have an alternative problem.

Comparison to Base The agent's learning curve can be seen in Figure 6.12a, this agent's learning curve, and it highlights a serious issue with its performance. It can be seen in this graph that this particular agent performs similarly to entropy-shunning agents, as it has improved rewards and sample efficiency. However, this does not remain the case for long, and instead, the reward quickly collapses at approximately forty-thousand episodes. Eventually, this results in it performing worse than the base approach. Based on Figure 6.13a, we posit that this may be due to the "moving agent" problem. This situation arises when one agent learns a good policy, while the other agent continues to explore or tries to improve its performance, potentially causing the

game to end prematurely. For instance, an agent that learns to emerge from behind a shield to shoot rapidly may put itself at risk, yet it gains a larger reward for doing so. Since rewards are not shared, a more cautious agent may see its rewards decrease as it fails to leave cover, shoot, and return to cover effectively. Moreover, learning an optimal strategy in such a game is challenging, as an agent can act greedily to obtain greater short-term rewards, while denying its teammate the opportunity to contribute to winning the game. During the training process, we observed that agents often engaged in exploratory behaviour that led to adopting a more reckless policy in order to gain rewards similar to their more "selfish" ally, who sacrifices the game early for the sake of quickly eliminating enemies.

In some environments, increasing non-stationarity can lead to a reduction in the overall team reward. Hence, in Space Invaders we may prefer the alternative of these entropy-embracing agents. Instead, we would prefer (as shown in Figure 6.3 with the best values for each approach) entropy-shunning agents in situations where we rely on our teammates. We come to this conclusion as, it can be okay if one player may perform slightly better than another, so long as both agents can view each other as reliable teammates who they can plan around. This can be seen with the values $\xi = -0.001, \kappa = 20$ in Figure 6.13b which yield the best overall performance in the Space Invaders environment despite the fact that the second agent contributes slightly more to the total score. However, there can be circumstances where one agent does not perform well and hence this imbalance can be an issue as seen in Figure 6.13b. Here, the values $\xi = -0.001, \kappa = 20$ lead to the best overall performance for RECO(neg) however, it also lead to the largest imbalance in agent performance. In such a case the "lazy" agent problem mentioned in Section 3.3.1 may reemerge. Therefore, a wide range of hyperparameters may be required when testing to find a set which suffers to a lesser degree from both the "moving" and "lazy" agent problems.



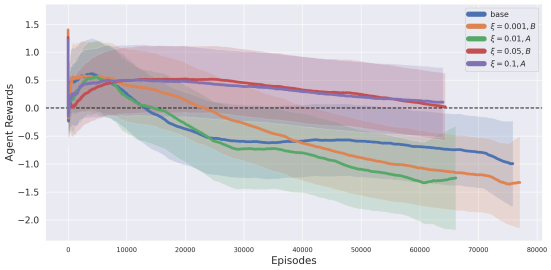
(a) RECO

(b) RECO(neg)

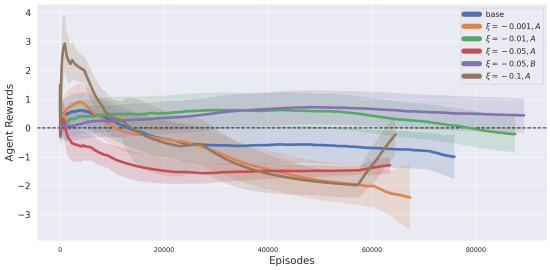
Figure 6.12: Performance of RECO and RECO(neg) in Space Invaders during training (higher is better). Agents can have a different number of steps in each episode (as outlined in Section 5.1) hence, although they train for the same length of time they have experienced the same number of steps.

6.3 Sensitivity of our Approaches

To illustrate how sensitive our entropy-shunning approaches can be to the values of the hyperparameters chosen, we do the following. We demonstrate how ADE(neg) and



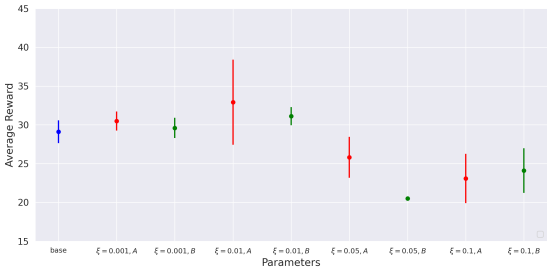
(a) RECO



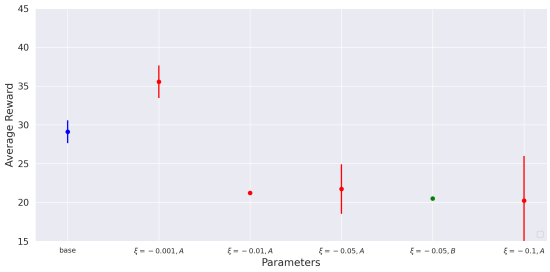
(b) RECO(neg)

Figure 6.13: Difference between the first agent and second agents rewards in the Space Invaders environment, with negative indicating 2nd agent has more influence (closer to zero is better). Agents can have a different number of steps in each episode (as outlined in Section 5.1) hence, although they train for the same length of time they have experienced the same number of steps.

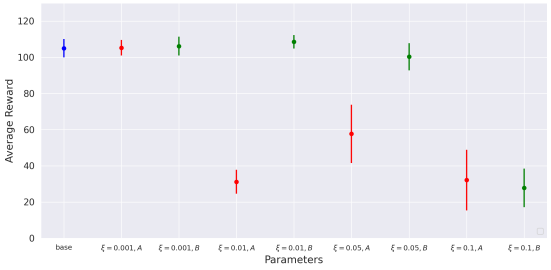
RECO(neg) perform in each environment and their sample efficiency. This demonstration will allow us to illustrate the need for careful hyperparameter tuning for these types of agents. Furthermore, we provide a table showcasing how many approaches can finish the full set of steps, further motivating the requirement for testing a wide range of hyperparameters.



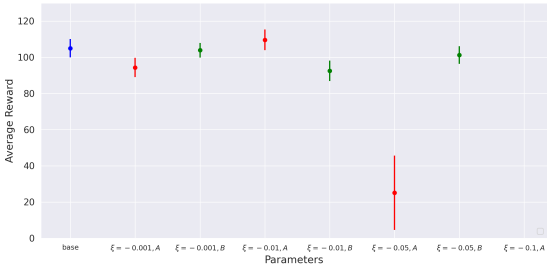
(a) Space Invaders ADE(neg)



(b) Space Invaders RECO(neg)



(c) Cooperative Pong ADE(neg)



(d) Cooperative Pong RECO(neg)

Figure 6.14: Mean performance and confidence intervals during the course of training for entropy-shunning agents and their various hyperparameter values compared to the base approach (higher means are better).

Figure 6.14, shows the mean reward and confidence intervals for various hyperparameter values for our entropy-shunning agents. We note that in Figure 6.14d, we see that the value we clip on for RECO(neg) can have extreme effects on agent performance. Here, for the same ξ value of 0.05, the difference between clipping on 2 instead of

20 can cause a dramatic decrease in agent performance, and increase the agent's variance. This problem is also illustrated with ADE(neg), where we see the same degree of sensitivity for the same values of ξ and κ in Figure 6.14c.

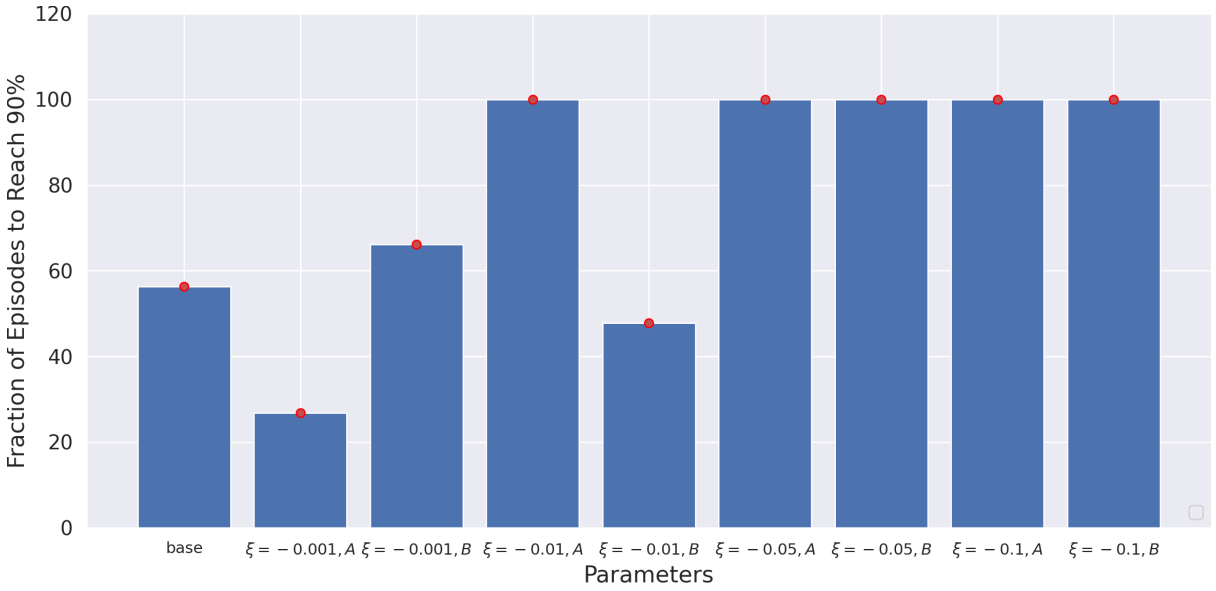


Figure 6.15: The fraction of episodes required by RECO(neg) to reach the 90^{th} percentile of rewards for the base approach in Simple Spread (lower is better). Here, approaches which reach the 100^{th} percentile never reach the same performance of the base algorithm.

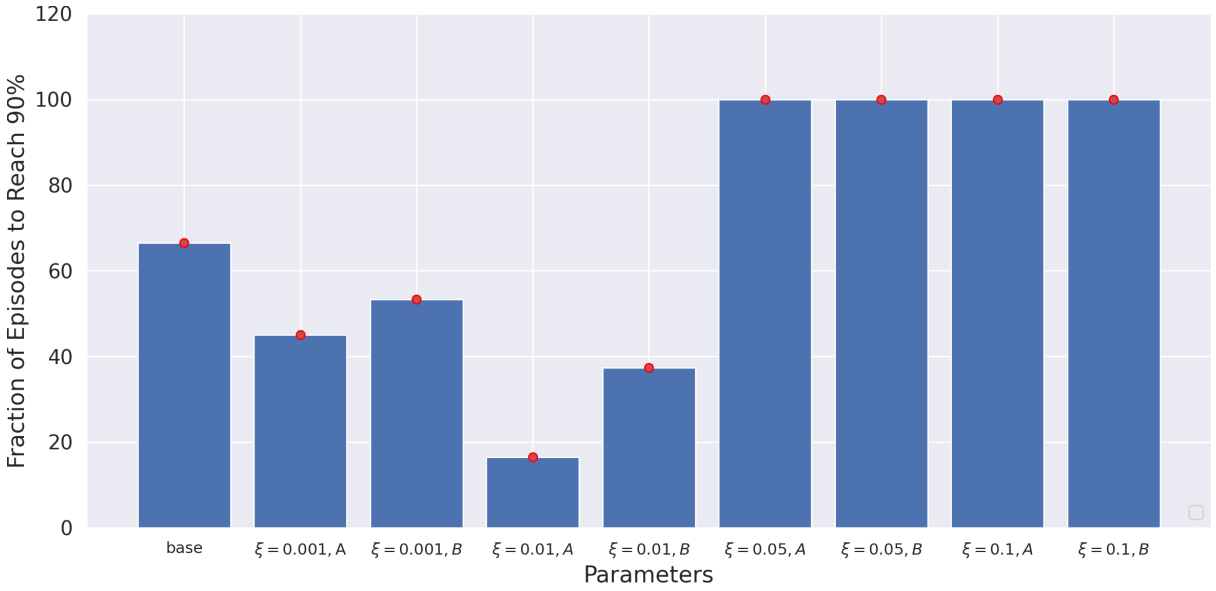


Figure 6.16: The fraction of episodes required by ADE(neg) to reach the 90^{th} percentile of rewards for the base approach in Space Invaders(lower is better). Here, approaches which reach the 100^{th} percentile never reach the same performance of the base algorithm.

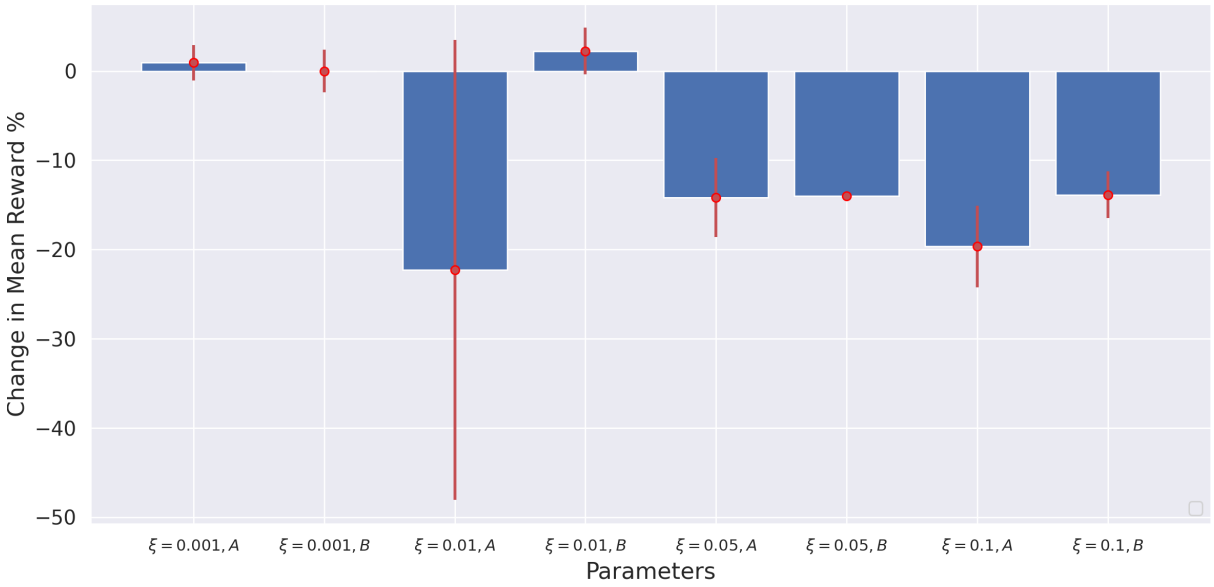


Figure 6.17: The difference in mean reward throughout training for various hyperparameter values of ADE(neg). Here, the reward is measured as a percentage increase or decrease compared to the base approach, and it contains the confidence intervals (higher value are better).

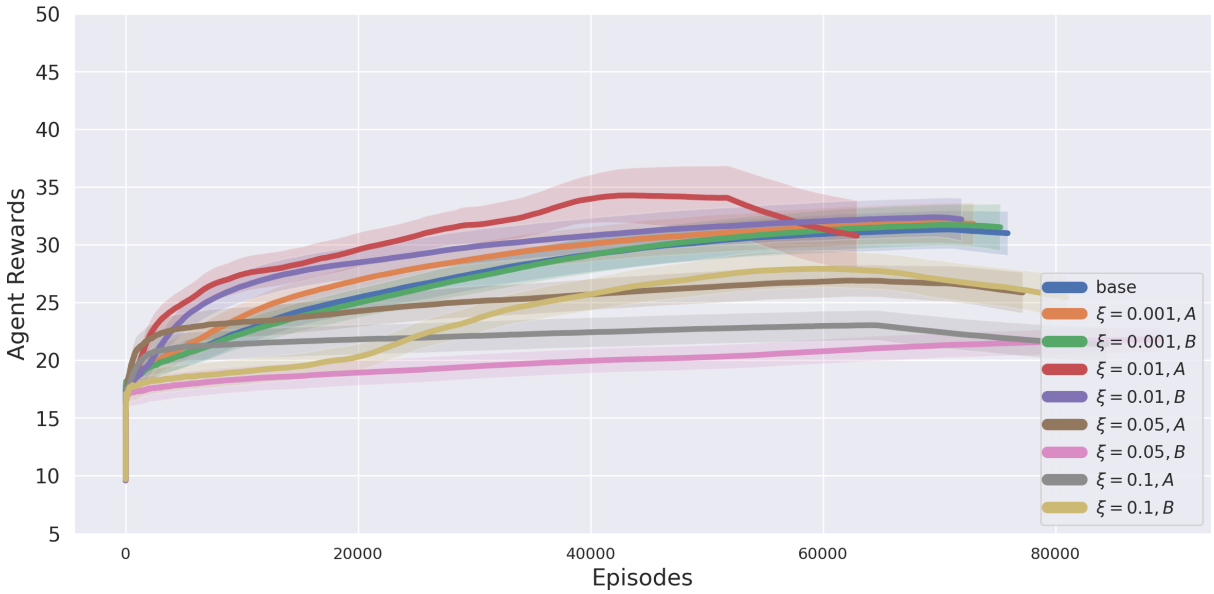


Figure 6.18: Performance of ADE(neg) in Space Invaders during training. Agents can have a different number of steps in each episode (as outlined in Section 5.1) hence, although they train for the same length of time they have experienced the same number of steps.

Furthermore, clear cases exist where only one set of hyperparameters allows the agent to learn. Figure 6.15 shows that in the Simple Spread environment, one of the RECO(neg) approaches ($\xi = 0.001, \kappa = 20$) can outperform the base method by requiring nearly half the number of episodes. The other results of the hyperparameter settings of

RECO(neg) are incapable of reaching the same performance as the base approach. Hence, this approach can be highly sensitive to hyperparameters and require fine-tuning. Similarly, we see for ADE(neg) in Figure 6.16 that ADE(neg) can outperform the base approach for some hyperparameter values. However, upon closer inspection, we can see in Figure 6.17 that while there may be some increased sample efficiency, the total performance of this method can dramatically vary. In particular, only one set of values ($\xi = 0.01, \kappa = 2$) truly outperforms the base approach regarding the mean reward over the training period. Therefore, to demonstrate the various parameters rewards throughout episodes we use Figure 6.18. This figure indicates that only two of the hyperparameter value sets are capable of being meaningfully more sample efficient.

6.3.1 Number of Failures

Lastly, we provide the following table to illustrate how often the various approaches for entropy-shunning and entropy-embracing agents resulted in agents failing to learn due to the hyperparameter sets causing various issues. Namely, in this case, the issues would result from encountering *NaNs* (not a number) due to exploding or vanishing gradient issues. Whilst this is an important issue, we relate to it further in our discussion, as it exists as a potentially open problem with our method. We use eight different sets of hyperparameters for each approach when tuning (as shown in Table 5.2). Additionally, we test five approaches for each environment, resulting in a total of forty experiments per environment.

Environment	Shannon	ADE	ADE(neg)	RECO	RECO(neg)	Total Successes
Simple Spread	4	5	7	**2**	5	23
Space Invaders	4	**2**	3	**2**	**2**	13
Cooperative Pong	8	8	7	7	**6**	36

Table 6.1: Table illustrating the number of times each approach was able to succeed in training the full number of steps in a given environment. Bold numbers indicate which methods had the lowest success rate. In each case, each approach had eight sets of hyperparameters tested. Therefore, there are forty experiments per environment (for all approaches with their respective hyperparameters)

In Table 6.1, our entropy-shunning agents typically fail slightly less when training than their counterparts. Although, it is essential to note that, in general, both sets of agents can be prone to fail in particular circumstances. We also note that certain environments are more prone to failure than others. In particular, Space Invaders proves to be the most challenging with one-third the number of successes compared to Cooperative Pong.

6.4 Ablations

In our research, we aim to investigate the impact of several design choices on the performance of our approach, as demonstrated in Algorithm 2. Our inquiries include the

consequences of setting the base ζ to zero, which modifies the base entropy used in the algorithm, the impact of clearing the accumulated entropy during training on the agents' performances, the significance of changing the additional entropy bonus on performance, and the impact of modifying log probabilities stored by the PPO algorithm on the agents' performances. While our approach relies on the base ζ term to function effectively in some cases, omitting it can have potential negative consequences. Additionally, we explore the impact of clearing the accumulated entropy, which may seem counterintuitive but is effective during training. The clearing of this entropy would result in the agent using only recent entropy accumulated during the policy rollout phase instead of the running entropy during the entire training period. We also investigate the significance of changing the additional entropy bonus on the performance of our approach. Furthermore, we examine the impact of modifying log probabilities on performance, which has been found to improve the performance of our approach in some tasks, notably in Space Invaders. Overall, our investigations aim to demonstrate the necessity of these design choices when conducting our experiments. We provide the following concise summary of each modification:

- All: This approach employs all the modifications described below.
- Base Zeta (BZ): This modification sets the base ζ term to zero, diverging from the value we set of -0.17. We set our entropy value to have a default value of -0.17 which gets added with the entropy during policy roll out and evaluation.
- Clearing entropy (CLR): In this case, the accumulated entropy during training is cleared. We solely utilise the most recent entropy, specifically the entropy accumulated during the policy rollout. After updating the policy, the entropy is cleared once again.
- EB: With this approach, there is no modification to the entropy added to the policy loss function. Instead, the effect is limited to the surrogate function through this additional source of entropy.
- SR: While this method does not modify the log probabilities stored during the policy rollout, it does alter the entropy added to the policy loss function.

In general, we observe that the performance of all approaches when employing all modifications does not significantly hinder any agent. Although there are two instances where the performance varies more substantially, these deviations are not large enough to create a severe disadvantage when comparing our primary focus in this dissertation, which is examining the effects of negative entropy (shunning entropy) instead of embracing it. Our first example of this change is illustrated by Shannon entropy and ADE in Table 6.2. Here, performance decreases by approximately 10% when not using a base zeta value, clearing entropy, and not modifying the surrogate function. Nevertheless, even in the best case for this particular value, our top-performing method (RECO(neg)) outperforms both of these approaches, even if they were to employ the optimal modifications for these approaches under this specific circumstance. RECO(neg) would have surpassed this value by 28.47%, and it demonstrated a faster learning rate, which aligns with one of the primary goals of entropy regularisation.

In the other cases, such as those in Tables 6.3 and 6.4, we observe that all approaches with each modification lie within one standard deviation of each other in terms of their final performance. We offer more detailed information for each ablation and its impact on learning in our Appendix, where the learning curves (similar to those in Figures 6.1, 6.3, and 6.7) depict the agents' performance during training.

Mod	Shannon	ADE	ADE(neg)	RECO	RECO(neg)
All	28.12 ± 2.75	28.11 ± 2.75	32.28 ± 3.08	30.27 ± 2.58	$\mathbf{39.87 \pm 3.01}$
BZ	28.65 ± 2.21	28.65 ± 2.21	$\mathbf{32.83 \pm 2.67}$	30.18 ± 2.58	29.21 ± 3.93
CLR	28.14 ± 2.52	28.14 ± 2.58	32.75 ± 2.67	30.17 ± 2.54	$\mathbf{34.56 \pm 4.20}$
EB	28.11 ± 2.27	28.11 ± 2.27	32.61 ± 2.58	30.16 ± 2.32	$\mathbf{35.83 \pm 3.39}$
SR	31.04 ± 2.47	31.04 ± 2.47	31.19 ± 2.61	$\mathbf{31.38 \pm 2.80}$	30.73 ± 3.02

Table 6.2: This table displays the mean reward and standard deviation in the Space Invaders environment over the final 1000 episodes for each approach using various modifications. The approach which has the best mean performance is indicated in bold.

Mod	Shannon	ADE	ADE(neg)	RECO	RECO(neg)
All	-40.08 ± 1.31	-39.59 ± 1.38	$\mathbf{-38.78 \pm 1.38}$	-38.80 ± 1.37	-40.53 ± 1.81
BZ	-40.82 ± 1.67	-39.49 ± 1.41	-39.19 ± 1.42	$\mathbf{-38.90 \pm 1.37}$	-39.07 ± 1.37
CLR	-39.78 ± 1.35	-39.43 ± 1.43	-38.96 ± 1.40	$\mathbf{-38.82 \pm 1.39}$	-39.72 ± 1.60
EB	-39.67 ± 1.31	-39.70 ± 1.57	$\mathbf{-38.85 \pm 1.36}$	-39.00 ± 1.43	-40.16 ± 1.65
SR	-38.8 ± 1.40	-38.99 ± 1.43	-38.81 ± 1.40	$\mathbf{-38.76 \pm 1.38}$	-39.35 ± 1.37

Table 6.3: This table displays the mean reward and standard deviation in the Simple Spread environment over the final 1000 episodes for each approach using various modifications. The approach which has the best mean performance is indicated in bold.

Mod	Shannon	ADE	ADE(neg)	RECO	RECO(neg)
All	179.96 ± 29.20	179.25 ± 28.14	183.15 ± 27.53	179.72 ± 28.52	$\mathbf{185.44 \pm 30.02}$
BZ	180.12 ± 30.17	178.62 ± 28.88	179.14 ± 28.55	176.47 ± 27.68	$\mathbf{183.49 \pm 30.92}$
CLR	176.57 ± 27.62	176.21 ± 27.64	180.52 ± 29.62	176.16 ± 27.38	$\mathbf{183.87 \pm 28.26}$
EB	178.75 ± 28.90	176.07 ± 27.54	180.87 ± 29.87	177.40 ± 27.21	$\mathbf{182.96 \pm 28.50}$
SR	179.85 ± 28.94	180.19 ± 29.75	180.52 ± 29.39	$\mathbf{181.99 \pm 30.03}$	177.48 ± 28.39

Table 6.4: This table displays the mean reward and standard deviation in the Cooperative Pong environment over the final 1000 episodes for each approach using various modifications. The approach which has the best mean performance is indicated in bold.

6.5 Summary

In this chapter, we have showcased the following. We demonstrated in Section 6.1 that our entropy-shunning agents can achieve at least the same rewards as the base

in the worst case. In their best case these agents are capable of doing better than the base approach in terms of rewards, sample efficiency or potentially both. Furthermore, our entropy-shunning agents outperform the entropy-embracing agents, which follow the standard approach of encouraging exploration by using entropy as a reward. Our experiments using entropy-shunning agents demonstrate that they can achieve similar outcomes to those traditionally achieved by entropy-embracing agents, which are known to improve sample efficiency and potentially enhance rewards. We illustrated in Section 6.2 that increasing the entropy reward in fully-cooperative, fully-decentralised MARL can lead to increasingly worse agent performance. Finally, we highlighted one crucial concern in our method in Section 6.3. In particular, this concern showcases the sensitivity of our method to the combination of its hyperparameters and hence, demonstrates the need to search over a wide enough range of hyperparameters. We discuss potential solutions that should be investigated to mitigate this problem in our conclusion. Additionally, we provide a full set of figures for each of the graphs shown in this chapter in our appendix to provide a holistic view of all approaches' performances.

Chapter 7

Competitive Results and Discussion

In this section, we explore the competitive case, building upon our demonstration of the benefits of sometimes shunning entropy in cooperative scenarios. We investigate the performance of our approach in competitive environments, such as competitive pong and boxing, as detailed in Chapter 5. The primary motivation behind these experiments is to examine whether striving for more deterministic behaviour could potentially lead to a disadvantage in a competitive setting. To determine whether a reliable agent could be exploited by an unpredictable competitor, it is necessary to examine their respective degrees of predictability and flexibility. Naivety resulting from reliability could make it easier to plan against, especially given that both agents have memory (in the form of GRUs) to recognise such patterns.

We begin by providing an overview of our experimental setup in Section 7.1, discussing the need for fairness, the choice of hyperparameters, and the structure of our Round-Robin tournament. Next, we delve into the analysis of agent performance in this tournament in Section 7.2. We offer a concise summary of our findings in Section 7.3, aiming to determine the applicability of our approach in competitive cases. Lastly, for all figures, we use the same labelling for each set of hyperparameters as set out in Table 5.2. Hence, when clipping we use let $A = 20$ and $B = 2$.

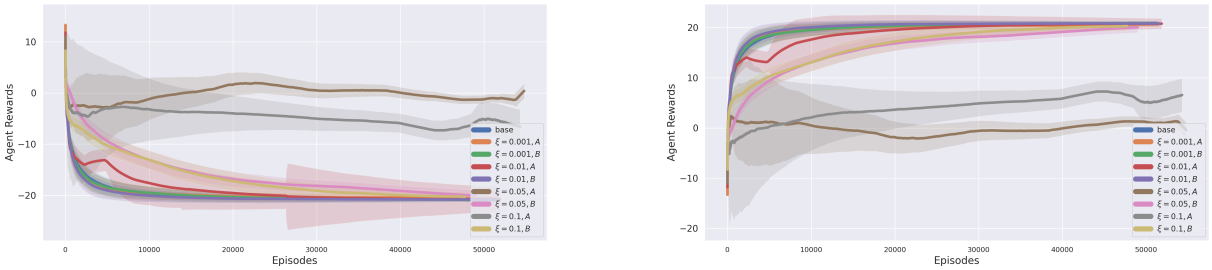
7.1 Tournaments

To determine whether our entropy-shunning agents perform differently from their alternative entropy-embracing counterparts, we perform the following empirical investigation:

1. Find desirable hyperparameters for each approach.
2. Create a Round-Robin tournament.
3. Explain the position bias and how we overcome it. Position bias refers to whether an agent plays as player one or as player two in a particular environment.

By doing this, we give each of the agents the best chance to perform. Hence, this allows us to demonstrate whether our method performs worse than its alternative. We do this as not all domains are cooperative; therefore, we must understand the potential impact of being more deterministic than an opponent in a competitive setting. Our hypothesis is that encouraging agents to be more deterministic would render them susceptible to their alternative counterparts, which embrace entropy. Consequently, the entropy-embracing agents would gain an advantage and outperform the deterministic agents by a large margin The following subsections explain each component of our experiment, its purpose and why they are necessary.

7.1.1 Fairness



(a) Agent one

(b) Agent two

Figure 7.1: Agents' rewards using the ADE(neg) algorithm in the Pong environment for each of the two agents.

In our evaluation, fairness is a crucial factor when deciding which hyperparameters to use for each agent, as we aim for a balance between agents in position one and position two. The importance of fairness stems from the fact that the best-performing agent is also the worst-performing agent in a zero-sum game. When an agent wins a one-sided game by the maximum amount, it indicates that the same set of hyperparameters resulted in an agent that lost completely. For example, when one set of parameters results in the agent scoring 20 as player two, these same parameters lead to the agent receiving a score of -20. This phenomenon is demonstrated in our results, as shown in Figures 7.1a and 7.1b, where the first and second agents' rewards for the Pong environment are presented, respectively.

To address this issue, we measure the distance between the rewards of the two agents to evaluate how a particular set of hyperparameters performs in a game or how two different approaches compare against each other. We calculate this by subtracting the first agent's rewards from the second agent's rewards, with a positive value indicating that agent one is winning, and a negative value indicating that agent two is winning. The value of this reward helps us determine how the game is fluctuating between the agents and which agent is dominating the game, or alternatively, how fair the game between agents is. Our goal is to find agents with rewards closer to zero, indicating a less one-sided game and a more "fair" game for the agents. We also consider the lengths of games to avoid selecting agents that merely prolong the game by doing nothing or lose too rapidly. We prioritise agents that exhibit fairness, meaning they

maintain a more balanced performance with rewards closer to zero, indicating a less one-sided game. Furthermore, we prioritise agents that demonstrate an increase in episode length during training, as this suggests they are actively engaging with one another while avoiding immediate losses.

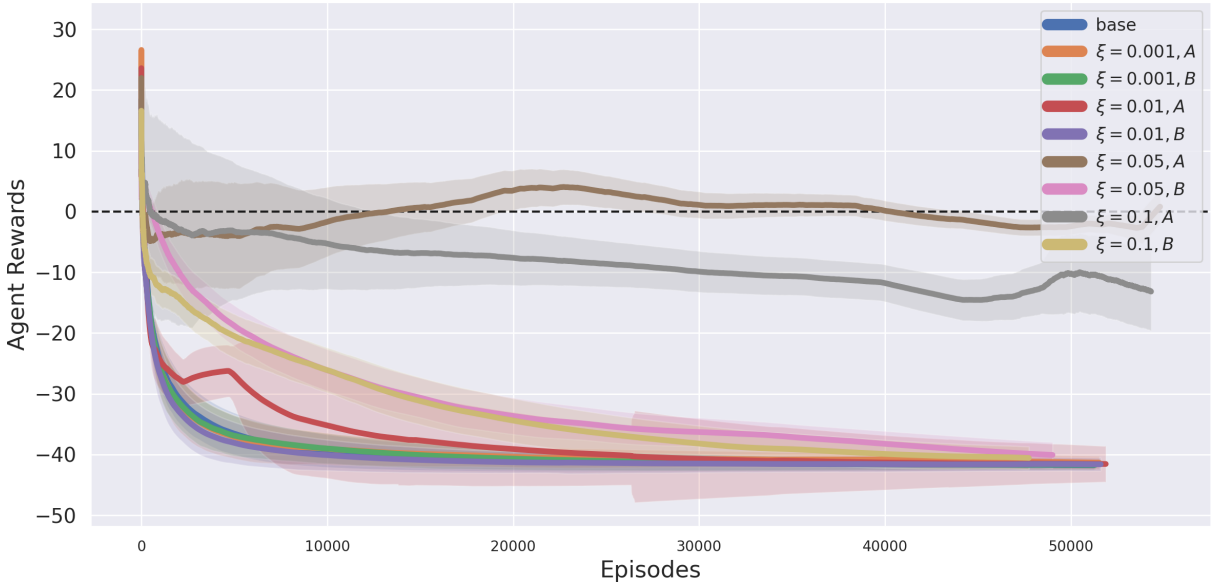


Figure 7.2: Distance between agent one's and agent two's rewards in Pong using ADE(neg). Here, closer to zero is preferred.

To demonstrate this, we illustrate the performance of one approach, its hyperparameters and the distance between agents' rewards in Figure 7.2. We selected this example because it exemplifies a scenario where certain approaches achieve a more equitable and balanced gameplay, reflected by two sets of hyperparameters ($\xi = 0.05, \kappa = 20$ and $\xi = 0.1, \kappa = 20$) with a mean score closer to zero during training. However, we also present the performance of other approaches in the Appendix. In this case, we can see that $\xi = 0.05, \kappa = 20$ and $\xi = 0.1, \kappa = 20$ where the difference between agents is the closest to zero, and hence, indicates a more fair game. Therefore, we choose agents similar to these for all approaches. However, we found that, in general, most games are not fair and have a strong position bias, where the position an agent plays (e.g. being player one) substantially impacts their reward. These results for the other approaches (such as ADE, Shannon entropy and RECO) are similar to the other hyperparameters (excluding $\xi = 0.05, \kappa = 20$ and $\xi = 0.1, \kappa = 20$) from Figure 7.2.

7.1.2 Parameters

Table 7.1 summarises the set of hyperparameters we chose for each approach. We chose these hyperparameters based on the idea of balance and fairness outlined earlier in Section 7.1.1. These hyperparameters are used for both the Pong and Boxing environments. Additionally, we have two sets of hyperparameters for ADE(neg) and RECO(neg), as indicated in Table 7.1. We do this because these agents had two sets of promising hyperparameters. Therefore, we use both sets of parameters to give each approach the best opportunity. It is important to note that the ADE(neg) and RECO(neg)

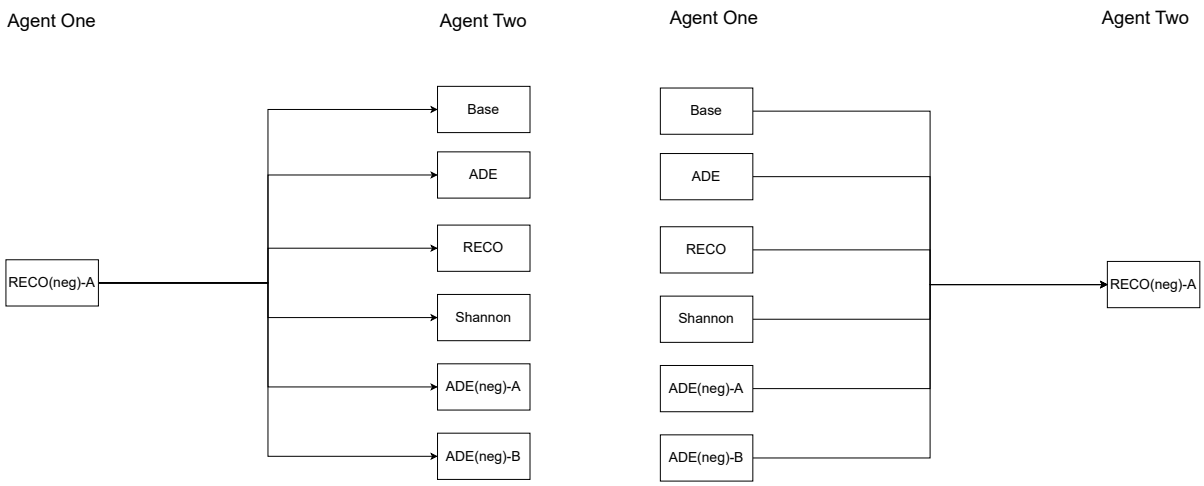
approaches each have two variations, denoted as A and B. To avoid potential biases caused by comparing different variations of the same approach (i.e. different hyperparameters but representing the same concept), we do not allow the A and B versions of either approach to compete against each other in our experiments. For example, ADE(neg)-A will not compete against ADE(neg)-B. However, the two variations of the ADE(neg) approach, denoted as A and B, are evaluated against all other approaches in our experiments, including both variations of the RECO(neg) approach, denoted as A and B. Therefore, we compare the performance of all four variations of these two approaches to determine which combination of hyperparameters performs the best. This practical choice is because our primary concern is how each approach competes against the other methods, not how the various hyperparameters compare when competing against themselves. This choice helps us simplify our results and convey the primary concern; is it disadvantageous to shun entropy when your opponent embraces it?

Approach	ξ	κ
Base	–	–
Shannon	-0.2	–
ADE	-0.2	–
ADE(neg)-A	0.05	20
ADE(neg)-B	0.1	20
RECO	0.1	20
RECO(neg)-A	-0.1	20
RECO(neg)-B	-0.01	20

Table 7.1: Table containing the hyperparameters used for each approach, respectively. In cases where the approach does not need a specific hyperparameter we use a dash to denote this.

7.1.3 Round-Robin

We use a Round-Robin tournament to evaluate each approach and determine how they perform against each other. To aid in our explanation, we provide Figure 7.3 to demonstrate the tournament structure for a given agent. In this tournament, each agent using a specific approach, such as RECO(neg)-A, will play against all other agents using various alternative methods. However, all methods share a similar structure and follow the same Round-Robin tournament format, ensuring that all agents play against each other. To do this, we refer to Figure 7.3a. First, the agent will play as player one, then it will play against the other approaches. Following this, the agent will become player two and play a second game against all other approaches, as shown in Figure 7.3b. Given the computational complexity of this setup, we do this with ten seeds. In practical terms, all these experiments took approximately two weeks to complete, running them in parallel using vector environments on sixteen computers. Additionally, structuring the tournament in this format helps us acknowledge the position bias, as both agents



(a) First phase

(b) Second phase

Figure 7.3: Figure illustrating how we structure the Round-Robin tournament in two phases to allow each agent to play as both player one and player two. Although we perform the same Round-Robin tournament for each agent, we only demonstrate the tournament from the perspective of RECO(neg)-A for the sake of simplicity in our illustration. Thus, the tournaments shown here only involve RECO(neg)-A.

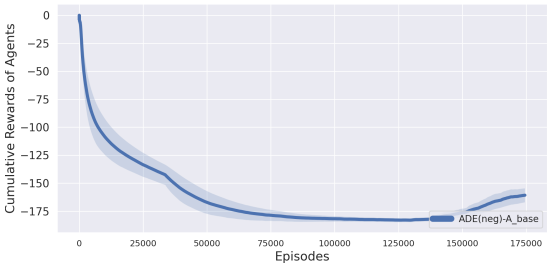
can play both sides. Hence, their overall performance will incorporate the benefit of this bias and its detriments.

7.2 Results

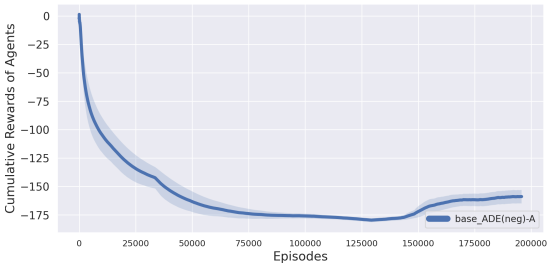
We notice some key trends when investigating all agents' performances for Boxing and Pong. Using entropy-shunning agents does not impact agents negatively. In general, agent performance between the entropy-shunning agents, entropy-embracing agents and the base approach is quite similar in performance. The most considerable impact on agent performance comes from the position of the agents. The position an agent plays has a greater importance on performance than the algorithm an agent uses. We provide an example of this result; however, the general trend remains for all various approaches. Lastly, we train each approach for the same number of steps however, we limit the training to this number for a particular reason namely, that both agents continue to learn so given the non-stationarity there is continual oscillation in the agent performance. For example, one agent may suddenly start to perform better than it previously was, or conversely worse than it was prior and then suddenly change. This fluctuating nature is largely due to the non-stationarity that exists with the two agents simultaneously learning in the same environment.

7.2.1 Position Bias

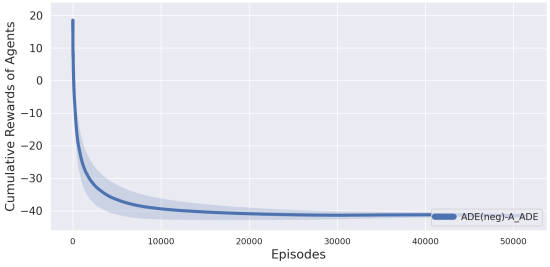
Figure 7.4 illustrates the effects of positional bias in both Boxing and Pong environments. Figures 7.4a and 7.4b depict ADE(neg)-A competing against the base approach,



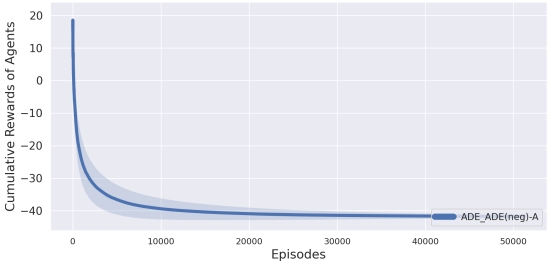
(a) ADE(neg)-A versus the base approach



(b) The base approach versus ADE(neg)-A



(c) ADE(neg)-A versus ADE



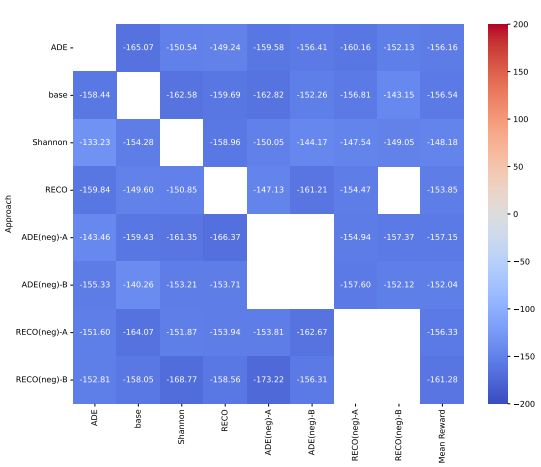
(d) ADE versus ADE(neg)-A

Figure 7.4: The positional bias in Boxing and Pong can be seen in these examples. The upper row shows examples from Boxing, while the bottom row shows examples from Pong. Each subcaption indicates the competing approaches, with player one mentioned first, followed by player two. These examples illustrate that in both games, player one is at a disadvantage to player two, regardless of the approach they use.

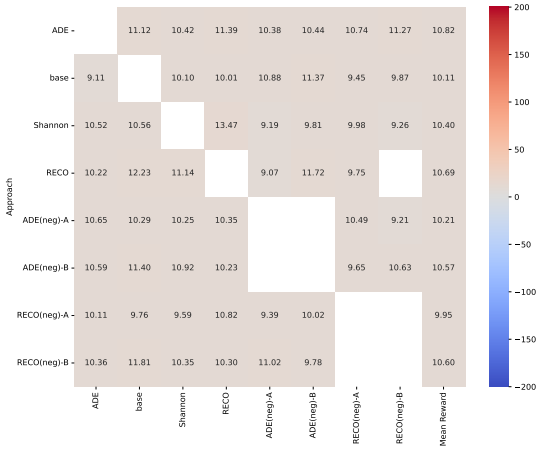
with Figure 7.4a representing ADE(neg)-A as the first agent and Figure 7.4b as the second agent. In both scenarios, position strongly influences agent performance, as both sets of agents lose when they are player one. Similarly, Figures 7.4c and 7.4d demonstrate this effect in the Pong environment, with ADE(neg)-A training against ADE. The positional bias is evident in this domain as well. Moreover, we observe that in the Boxing domain, although a strong position bias persists, agents avoid losing by the maximum amount, as seen in Pong. The slight increase in rewards towards the end of the training suggests that agents continue attempting to learn, yet remain unsuccessful in defeating their opponent. Our results where we find that one agent does much more poorly than another line up with those of Terry *et al.* [2022] and Lee *et al.* [2022].

Figure 7.5 presents the performance of each approach in the Boxing environment, where the distance between two agents' rewards for each game is recorded to emphasise the performance gap. Figures 7.5a and 7.5c display the mean rewards for the approaches when playing as player one and player two, respectively. Notably, when an agent is player one, they consistently lose, but the extent of the loss varies. Shannon entropy experiences the smallest loss when playing as player one. Interestingly, the second-best performing approach, ADE(neg)-B, which shuns entropy, outperforms its alternative, ADE, although their mean standard deviations are closely related, as illustrated in Figure 7.5b. Moreover, we observe in Figure 7.5a that the RECO(neg)-B method performs the poorest in this context, achieving a mean reward of -161.28 against other approaches. However, all approaches exhibit similar mean standard deviations, as depicted in Figure 7.5b.

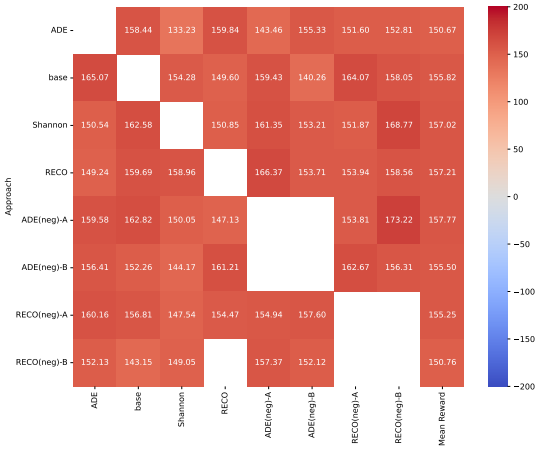
Figure 7.5d shows agent performance when each approach plays as the second player. In this situation, all approaches prevail, but the winning margin varies. ADE emerges as the winner with the smallest margin against Shannon entropy. In addition, ADE(neg)-A and Shannon entropy achieve two of the highest scores against another approach when playing against RECO(neg)-A. Nevertheless, the score difference between the mean performance of each approach when an agent is player two remains relatively similar. Based on the results in Figure 7.5, we identify two trends. First, the position bias exerts a substantial influence on agent performance, which we investigate further in Section 7.2.3 and incorporate the results from Pong. Second, each approach appears to be vulnerable to a specific approach, such as ADE competing against Shannon entropy or ADE(neg)-A competing against RECO(neg)-B.



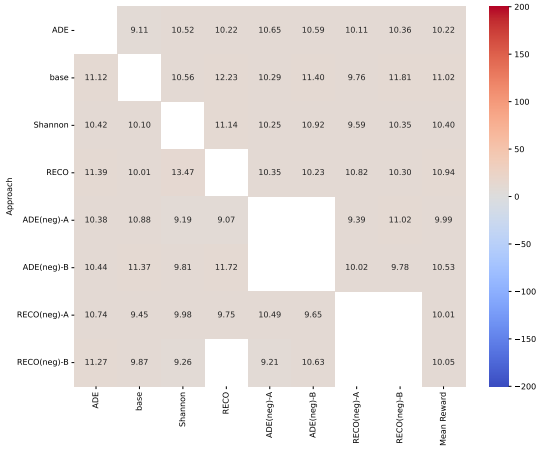
(a) Player One Rewards



(b) Player One Standard Deviation in Rewards



(c) Player Two Rewards



(d) Player Two Standard Deviation in Rewards

Figure 7.5: Final performance for the last 10% of episodes in Boxing for all approaches. The rewards and standard deviation are measured using the distance metric mentioned earlier in Section 5.3 however, each agent's individual reward would be half the reward illustrated above. These graphs demonstrate the mean rewards and standard deviation in these rewards when each approach plays as player one and two respectively.

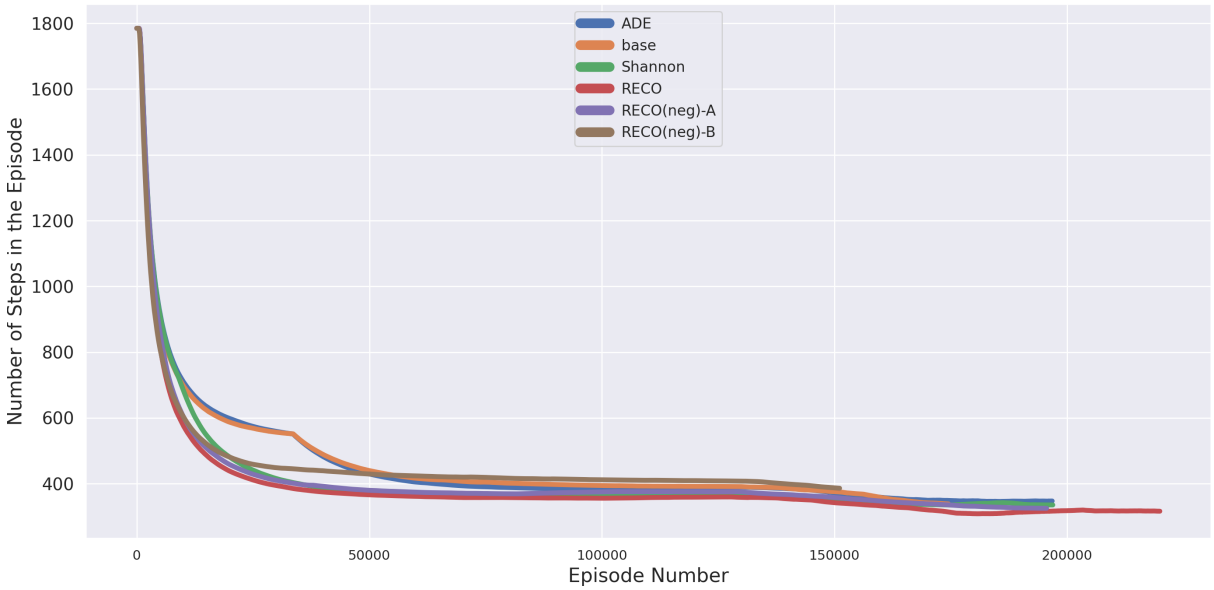


Figure 7.6: Lengths of each game in Boxing for ADE(neg)-A against all other approaches in the Round-Robin tournament. Each approach has the same number of total steps to train over however, as illustrated by the base approach, the number of steps in an episode may vary. Therefore, the lines can vary in terms of the number of episodes.

In the study conducted by Lee *et al.* [2022], various reinforcement learning algorithms, such as PPO, MAPPO, DQN, and others, were employed, allowing these algorithms to compete against one another. The researchers facilitated agent interaction as both players in Boxing and Pong, while also enabling agents to share observations through parameter sharing. For the Boxing environment, they discovered that agents achieved more victories when playing as the second player, demonstrating a substantial bias towards this position. In contrast, their investigation of Pong revealed that it was more challenging for the second agent to secure a win. Despite utilising parameter sharing in their approach, which enables agents to view the game from the perspectives of both the first and second player, the position bias remained a significant issue. Terry *et al.* [2022] also examined the performance of parameter sharing in Pong. In their study, they implemented multiple modifications to the observation space for the agents. Nevertheless, they found that the position bias posed a considerable challenge for the agent. Regardless of the observation modification, agents obtained similar rewards, with most averaging around -20. Consequently, learning in MARL may be particularly problematic, as Lee *et al.* [2022] observed that PPO's sensitivity as a method could be exacerbated in the MARL setting.

7.2.2 Game Lengths

As the game progresses, we found that the lengths of the games in terms of the number of steps become shorter as the training continue. Figure 7.6 demonstrates this trend. Initially, agents play relatively long games; however, as the training continues, agents play shorter games. Although Figure 7.6 represents the length of games in Boxing, we

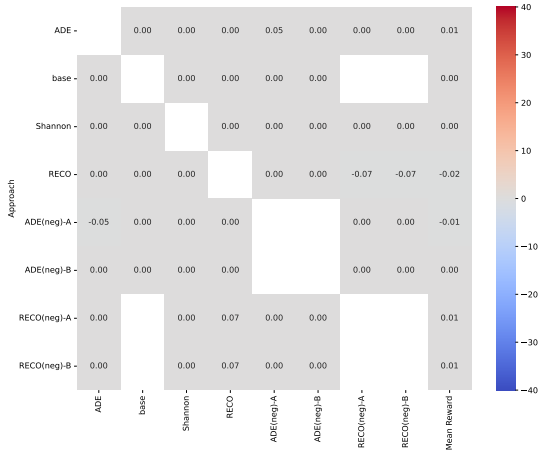
found the same trend existed for Pong, which we have included in the appendix. We found this to be the case for the other approaches; hence, this example is true for most Round-Robin games. Thus, position bias may be more problematic than the approach used.

7.2.3 Near End of Training Performance

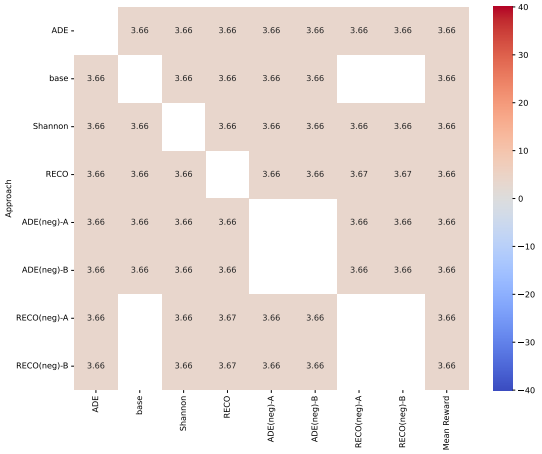
To demonstrate overall performance for all agents we construct heatmaps shown in Figure 7.7. These heatmaps represent the final 1000 episodes of each game for each of the environments in our Round-Robin tournament. We add both sets of rewards for when the agent was player one and player two. Then we take the mean of this score over the final 1000 episodes. This allows us to determine whether there is any notable performance difference between the various methods by the end of training. Whilst we compare each agent to the various other approaches, as mentioned in Section 7.1.3, we do not compare them to themselves (e.g. RECO(neg)-A will not be compared to RECO(neg)-A or RECO(neg)-B).

In Figures 7.7a and 7.7b, we present the final performance in the Pong environment. During our Round-Robin experiments, certain cases involved agents encountering errors, particularly when one agent in the tournament faced *NaNs*. Off-diagonal blank spaces in the results represent these instances. The white spaces on the diagonal indicate scenarios where the opponent agent used the same approach, which were not evaluated in the Round-Robin tournament, as explained earlier. In this environment, agents could achieve a *maximum distance* of rewards of 40 and a *minimum distance* of rewards of -40. However, the difference between various approaches (Figure 7.7a) is zero or approximately zero, with negligible standard deviation (as all approaches have near identical standard deviation), as seen in Figure 7.7b. This suggests that agents participated in fair games, but in reality, the position bias determined the winning player. Therefore, this environment indicates that in some circumstances, the approach used may not be important as the position of plays will ultimately determine the winner.

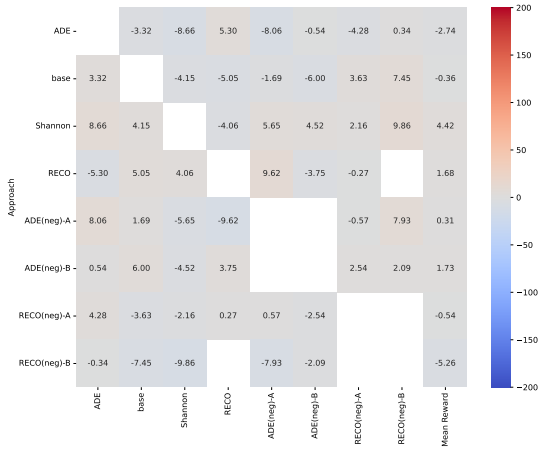
In contrast, Figures 7.7c and 7.7d present the performance of various approaches in the Boxing environment, where agents can achieve a *maximum distance* of rewards of 200 and a *minimum* of -200. The position bias continues to be influential, yet there is more variation in agent performance. In this environment, the agent with the lowest mean performance was RECO(neg)-B, registering a mean reward of -5.26. However, when considering both sides of the game, no agent managed to reach the maximum difference of 200 or even a value greater than 10 when competing against other approaches. This suggests that while agents' performance may exhibit slight variations against different approaches, the mean score remains close to zero, indicating minor overall differences once agents' performances as both player one and player two are considered. Moreover, Figure 7.7d reveals that agents display similar degrees of variation in their mean rewards when accounting for performance in both positions. Consequently, by considering both sides of the Round-Robin tournament for each agent, we find that, in general, the various approaches are quite similar and primarily diverge when competing against a specific approach. However, no substantial difference exists between them overall.



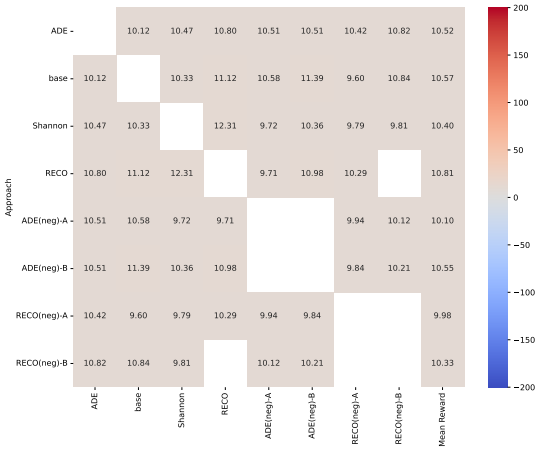
(a) Mean Rewards Pong



(b) Standard Deviation in Rewards Pong



(c) Mean Rewards Boxing



(d) Standard Deviation in Rewards Boxing

Figure 7.7: Heatmap of the mean reward and standard deviation for these rewards for the final 10% of training for each approach against its opponent in the Round-Robin tournament. A mean reward of zero indicates that the agents are equally matched.

7.3 Summary

In summary, we have demonstrated that our entropy-shunning agents do not experience a substantial disadvantage in competitive scenarios. As shown in Section 7.2.1 using Boxing as an example, agents' performances can vary to some extent when compared to other approaches. However, the degree of variation is not so extreme that one approach dominates all others. Instead, as we discovered in Section 7.2.3, when considering an agent's performance as both player one and player two, all approaches can exhibit similarities. Specifically, in the case of Pong (as depicted in Figure 7.7a), the particular approach employed may have no impact on the outcome, whereas the position plays a decisive role. On the other hand, in Boxing (as shown in Figure 7.7c), although agent performance did vary when examining their roles as player one and player two, when we accounted for their performance in both roles, the differences were still present but relatively small. Thus, we conclude that certain approaches may

perform better against others, but not to such an extent that they could still prevail when playing from a position disadvantaged role.

Chapter 8

Future Work and Conclusion

In this dissertation, we addressed an open problem in MARL concerning the functionality of multiple methods. Recent research on MARL has predominantly focused on the CTDE case rather than the fully-decentralised case, as noted by Gorsane *et al.* [2022]. This issue is pertinent due to potential bandwidth constraints or unavailable simulators, as highlighted in Chapter 3. Consequently, we concentrated on the fully-decentralised case, with minimal modelling of other agents or handcrafted methods. By foregoing handcrafted methods and modelling other agents, our methods become more broadly applicable to various domains or novel problems.

We proposed a novel approach that penalises entropy to tackle the challenge of non-stationarity in fully-decentralised MARL. This issue arises because the environment and other agents continually evolve, making it difficult for agents to learn effective policies. Our experiments revealed that penalising entropy achieves similar objectives as using reward entropy in single-agent RL, such as improving sample efficiency and potentially enhancing an agent's final rewards. Our approach to penalising entropy, rather than rewarding it, benefits agents in fully-decentralised MARL by improving their reliability to one another. This is in contrast to traditional methods that reward entropy to encourage exploration, which can exacerbate the ringing agent problem caused by non-stationarity. By subtracting entropy from the objective function, our approach effectively addresses non-stationarity by leveraging the stochastic nature of agents' actions, inherent in neural networks, instead of adding extra exploration through reward entropy. This key difference sets our approach apart from existing methods. Furthermore, we explored the use of alternatives to the current measure of entropy, such as RECO, which showed promising results in some cases, as demonstrated in our experiments with Space Invaders.

Our method surpasses the baseline (standard PPO without entropy regularisation) in games like Space Invaders and Cooperative Pong, while delivering performance akin to that in others, such as Simple Spread. Moreover, in Simple Spread (Figure 6.10e), we observed that increasing the entropy regularisation coefficient resulted in a subsequent decline in performance for both ADE and Shannon entropy. This renders our approach well-adapted for an array of cooperative tasks, illustrating its versatility and efficacy in diverse contexts. Our method may be particularly advantageous in real-world applica-

tions such as robotics or navigation systems, where agents possess limited communication and act autonomously. Our agents do not necessitate communication or information exchange during training, making them self-sufficient and suitable for autonomous systems that demand independent agents working towards a shared objective.

8.1 Limitations

Our future work's primary focus is to address the need for clipping and explore potential solutions, thereby simplifying hyperparameter tuning by eliminating one of the two hyperparameters in our method. The subtraction of entropy in our method introduces a clipping coefficient, necessitating additional hyperparameter tuning. Redesigning the equations, such as RECO(neg) and ADE(neg), might provide an alternative to clipping, initially intended to prevent exploding gradients. By circumventing the need for clipping, we could remove an extra parameter from the model. In some instances, our approach accumulates entropy, and the value may become excessively large, requiring further hyperparameter tuning to balance positive and negative rewards. Moreover, for methods like the standard actor-critic that lack a surrogate function, creating one could help agents avoid large policy changes, which some algorithms do not account for. This surrogate function is crucial because substantial policy updates may impair agent performance [Schulman *et al.* 2017b].

8.2 Future Work

We aim to investigate modifications and alternative modifications in greater detail in future work, examining their impact on various aspects of MARL and determining the most effective alterations for different scenarios. A potential modification might resemble that of Jaques *et al.* [2019], which includes an influence reward or that of Foerster *et al.* [2017b], which models other agents' expected policy updates. In line with our objectives, we could consider using a curiosity module similar to the work done by Pathak *et al.* [2017]. Rather than rewarding agents for influencing another agent, we could encourage them first to learn to model their environment, thereby aiding exploration. Following this, we might encourage agents to become more reliable as their predictions of the system's dynamics become more accurate. Hence, once agents can understand the dynamics, they are encouraged to become more deterministic and, thus, able to take their desired actions more regularly than would be the case if they were constantly encouraged to explore. Alternatively, we could apply the model to perform counterfactual reasoning as proposed by Foerster *et al.* [2018]. Agents could receive a difference reward based on what a model predicts would be their reward for taking the best possible action in the state rather than the action they actually take. This approach would enable agents to assign credit to themselves in a fully-decentralised manner without needing to model other agents. Moreover, it could integrate with our approach of encouraging the agent to be more reliable and, thus, more likely to select the perceived optimal action after adequate exploration.

We acknowledge that not all scenarios necessitate agents becoming increasingly deterministic from the outset. To achieve robustness akin to the maximum entropy demonstrated by Ziebart *et al.* [2008] and Haarnoja *et al.* [2018] in single-agent RL, we must adapt our approach to be more robust in cases where individual agents require some degree of exploration. An adaptive approach, incorporating alternating phases of entropy reward and reliability improvement, could prove beneficial. This adaptive approach would resemble the work of Xing *et al.* [2021], in which an agent's entropy usage changes during training. However, we would adapt it to allow an agent to independently decide the necessary entropy based on its circumstance rather than generating entropy for its teammates' policies. This flexibility may be particularly relevant in environments where initial exploration is vital due to reward sparsity. To enhance agents' reliability, we propose incorporating an action or state prediction module (i.e. a dynamics module similar to Pathak *et al.* [2017]) for each agent. As agents become more reliable, this module would enable them to better comprehend the system's dynamics, anticipate their teammates' actions, and predict environmental changes based on their current observation and what they can control. Consequently, agents can make more cooperative and forward-thinking decisions.

One promising direction for future research lies in examining the influence of entropy on inter-agent communication. For instance, previous studies such as that by Cui *et al.* [2022] have demonstrated the ability of MARL agents to establish meaningful communication protocols. Complex environments like Hanabi necessitate agents to devise strategies for conveying information regarding the potential actions of their counterparts at each step. We hypothesise that entropy-shunning agents might lean towards being more self-centered, especially considering the inherent noise when trying to establish a communication protocol. On the other hand, entropy-embracing agents might actively strive to develop such a protocol. This scenario emerges because agents aren't equipped with a predefined message-passing protocol and must cultivate one through interaction. However, constructing such a protocol demands extensive exploration, where the presence of entropy could prove beneficial by promoting diverse communication methods. In contrast, entropy-shunning agents might inherently shy away from experimenting with varied communication strategies. In scenarios demanding intricate communication techniques, experiencing a broad spectrum of information conveyance methods can be invaluable for enhanced coordination. Therefore, entropy-embracing agents might have the upper hand in achieving higher long-term rewards in such situations. Nonetheless, this hypothesis warrants further in-depth investigation.

In Section 6.1.2 concerning the Space Invaders environment, we observed that entropy-shunning was beneficial. In contrast, embracing entropy resulted in a decline in performance. Conversely, in Section 6.1.3 for Cooperative Pong, the impact of entropy was negligible, as all approaches performed similarly. This suggests a valuable research direction: determining which environmental attributes necessitate an agent to be rewarded for entropy. Or conversely, identifying environments where entropy should be avoided. Grasping when to prioritise high entropy policies — based on environmental attributes and when to eschew them would be invaluable in formulating an agent's policy and strategy. This topic merits consideration for future research.

8.3 Conclusion

Our research delineates the potential advantages of penalising entropy in the objective function within the realm of fully-decentralised MARL. Rather than underscoring the simplicity of our method, it's vital to highlight the pioneering nature of our approach. By venturing into the territory of penalising entropy, we introduce a fresh perspective into the established practices of MARL, offering an alternative to the prevailing paradigm that mostly champions the encouragement of entropy. Our method's adaptability is evident in its integration capability; by merely extending hyperparameter searches during training to encapsulate the regime of penalising entropy, our approach can seamlessly fit into existing frameworks, illustrating potential benefits in specific scenarios. Moreover, this adaptability showcases its practicality and potential applicability across a diverse range of contexts. Our exploratory efforts in the domain of negative entropy coefficients, in conjunction with positive coefficients, attest to our method's versatility.

In the cooperative environment, our findings indicate that our approach doesn't compromise on agent performance. In instances, it even excels beyond the baseline approach, notably outperforming the prevalent method of embracing entropy in games like Space Invaders. On the other hand, in competitive scenarios, our strategy of penalising agents based on their uncertainty doesn't significantly diminish their efficiency. Contrary to traditional practices that bolster entropy, our experiments with Simple Spread and, notably, Space Invaders suggest that such an approach might lead to deteriorating agent performance. Therefore, our alternative stance on entropy, advocating for its penalisation, emerges as a compelling, effective strategy, offering a fresh avenue to enhance the capabilities of current fully-decentralised MARL agents, especially in cooperative settings.

References

[Ahmed *et al.* 2019] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International conference on machine learning*, pages 151–160. PMLR, 2019.

[Albrecht and Ramamoorthy 2015] Stefano V Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *arXiv preprint arXiv:1506.01170*, 2015.

[Andrychowicz *et al.* 2020] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[Arulkumaran *et al.* 2017a] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[Arulkumaran *et al.* 2017b] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[Baker *et al.* 2011] Chris Baker, Rebecca Saxe, and Joshua Tenenbaum. Bayesian theory of mind: Modeling joint belief-desire attribution. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011.

[Baker *et al.* 2017] Chris L Baker, Julian Jara-Ettinger, Rebecca Saxe, and Joshua B Tenenbaum. Rational quantitative attribution of beliefs, desires and percepts in human mentalizing. *Nature Human Behaviour*, 1(4):0064, 2017.

[Bandura 1977] A. Bandura. *Social Learning Theory*. Prentice-Hall series in social learning theory. Prentice Hall, 1977.

[Bellman 1957] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[Bernstein *et al.* 2002] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[Botvinick *et al.* 2017] Matthew Botvinick, David GT Barrett, Peter Battaglia, Nando de Freitas, Darshan Kumaran, Joel Z Leibo, Timothy Lillicrap, Joseph Modayil,

Mohamed Shakir, Neil C Rabinowitz, et al. Building machines that learn and think for themselves. *Behavioral and Brain Sciences*, 40, 2017.

[Bowling and Veloso 2002] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

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

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

[Claus and Boutilier 1998] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[Cui *et al.* 2022] Brandon Cui, Andrei Lupu, Samuel Sokota, Hengyuan Hu, David J Wu, and Jakob Nicolaus Foerster. Adversarial diversity in hanabi. In *The Eleventh International Conference on Learning Representations*, 2022.

[Duan *et al.* 2016] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[Eysenbach and Levine 2019] Benjamin Eysenbach and Sergey Levine. If maxent rl is the answer, what is the question? *arXiv preprint arXiv:1910.01913*, 2019.

[Fernandes and BF 2017] Fernandes and Heitor BF. Darwin's unfinished symphony: How culture made the human mind. *Animal Behaviour*, 133:207–208, 2017.

[Foerster *et al.* 2017a] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 1146–1155. PMLR, 2017.

[Foerster *et al.* 2017b] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[Foerster *et al.* 2018] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[Foerster *et al.* 2019] Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951. PMLR, 2019.

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

[Gorsane *et al.* 2022] Rihab Gorsane, Omayma Mahjoub, Ruan de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *arXiv preprint arXiv:2209.10485*, 2022.

[Grover *et al.* 2018] Aditya Grover, Maruan Al-Shedivat, Jayesh Gupta, Yuri Burda, and Harrison Edwards. Learning policy representations in multiagent systems. In *International conference on machine learning*, pages 1802–1811. PMLR, 2018.

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

[Ha *et al.* 2016] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[Haarnoja *et al.* 2017] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

[Haarnoja *et al.* 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[Hansen *et al.* 2004] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, pages 709–715. AAAI Press / The MIT Press, 2004.

[Harari 2014] Yuval Noah Harari. *Sapiens: A brief history of humankind*. Random House, 2014.

[Hausknecht and Stone 2015] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[He *et al.* 2016] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *International conference on machine learning*, pages 1804–1813. PMLR, 2016.

[Hernandez-Leal *et al.* 2017] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[Hernandez-Leal *et al.* 2018] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Is multiagent deep reinforcement learning the answer or the question? a brief survey. *Learning*, 21:22, 2018.

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

[Herrmann *et al.* 2007] Esther Herrmann, Josep Call, María Victoria Hernández-Lloreda, Brian Hare, and Michael Tomasello. Humans have evolved specialized skills of social cognition: The cultural intelligence hypothesis. *science*, 317(5843):1360–1366, 2007.

[Hill *et al.* 2018] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. *Stable Baselines*. https://github.com/hill-a/stable-baselines, 2018.

[Hochreiter and Schmidhuber 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Horgan *et al.* 2018] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[Hsu *et al.* 2020] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. Revisiting design choices in proximal policy optimization. *arXiv preprint arXiv:2009.10897*, 2020.

[Hu and Foerster 2019] Hengyuan Hu and Jakob N Foerster. Simplified action decoder for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1912.02288*, 2019.

[Huang *et al.* 2022] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[Hughes *et al.* 2018] Edward Hughes, Joel Z Leibo, Matthew Phillips, Karl Tuyls, Edgar Dueñez-Guzman, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin McKee, Raphael Koster, et al. Inequity aversion improves cooperation in intertemporal social dilemmas. *Advances in neural information processing systems*, 31, 2018.

[Ilyas *et al.* 2018] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. *arXiv preprint arXiv:1811.02553*, 2018.

[Jaques *et al.* 2019] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*, pages 3040–3049. PMLR, 2019.

[Kanervisto *et al.* 2020] Anssi Kanervisto, Janne Karttunen, and Ville Hautamäki. Playing minecraft with behavioural cloning. In *NeurIPS 2019 Competition and Demonstration Track*, pages 56–66. PMLR, 2020.

[Kapturowski *et al.* 2019] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2019.

[Konda and Tsitsiklis 1999] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[Kullback and Leibler 1951] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[Laurent *et al.* 2011] Guillaume J Laurent, Laëtitia Matignon, Le Fort-Piat, et al. The world of independent learners is not markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 15(1):55–64, 2011.

[Lee *et al.* 2022] Ken Ming Lee, Sriram Ganapathi Subramanian, and Mark Crowley. Investigation of independent reinforcement learning algorithms in multi-agent environments. *Frontiers in Artificial Intelligence*, page 211, 2022.

[Leibo *et al.* 2017] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

[Liu *et al.* 2019] Jingbin Liu, Xinyang Gu, and Shuai Liu. Policy optimization reinforcement learning with entropy regularization. *arXiv preprint arXiv:1912.01557*, 2019.

[Lowe *et al.* 2017] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

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

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

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

[Myerson 1997] Roger B Myerson. *Game theory: analysis of conflict*. Harvard university press, 1997.

[Nash Jr 1950] John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[Nayyar *et al.* 2013] Ashutosh Nayyar, Aditya Mahajan, and Demosthenis Teneketzis. Decentralized stochastic control with partial history sharing: A common information approach. *IEEE Transactions on Automatic Control*, 58(7):1644–1658, 2013.

[Ng *et al.* 2000] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, page 2, 2000.

[Nguyen *et al.* 2020] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

[Oroojlooy and Hajinezhad 2022] Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, pages 1–46, 2022.

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

[Pathak *et al.* 2017] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[Peshkin *et al.* 2000] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. In *UAI*, pages 489–496. Morgan Kaufmann, 2000.

[Peysakhovich and Lerer 2017] Alexander Peysakhovich and Adam Lerer. Prosocial learning agents solve generalized stag hunts better than selfish ones. *arXiv preprint arXiv:1709.02865*, 2017.

[Premack and Woodruff 1978] David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526, 1978.

[Rabinowitz *et al.* 2018] Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick. Machine theory of mind. In *International conference on machine learning*, pages 4218–4227. PMLR, 2018.

[Rashid *et al.* 2020] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

[Roese 1997] Neal J Roese. Counterfactual thinking. *Psychological bulletin*, 121(1):133, 1997.

[Ross *et al.* 2010] Stéphane Ross, Geoffrey J Gordon, and James A Bagnell. *No-regret reductions for imitation learning and structured prediction*. Technical report, 2010.

[Rumelhart *et al.* 1986] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[Schaul *et al.* 2015] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[Schmidhuber 2010] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE transactions on autonomous mental development*, 2(3):230–247, 2010.

[Schulman *et al.* 2015a] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[Schulman *et al.* 2017a] John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

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

[Sequeira *et al.* 2011] Pedro Sequeira, Francisco S Melo, Rui Prada, and Ana Paiva. Emerging social awareness: Exploring intrinsic motivation in multiagent learning. In *2011 IEEE international conference on development and learning (ICDL)*, volume 2, pages 1–6. IEEE, 2011.

[Shannon 1948] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[Shoham *et al.* 2007] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial intelligence*, 171(7):365–377, 2007.

[Strens 2000] Malcolm Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, volume 2000, pages 943–950, 2000.

[Sunehag *et al.* 2017] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

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

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

[Terry *et al.* 2020a] Justin K Terry, Benjamin Black, and Ananth Hari. Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*, 2020.

[Terry *et al.* 2020b] Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020.

[Terry *et al.* 2021] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

[Terry *et al.* 2022] J. K. Terry, Nathaniel Grammel, Sanghyun Son, and Benjamin Black. *Parameter Sharing For Heterogeneous Agents in Multi-Agent Reinforcement Learning*, 2022.

[Toussaint 2009] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

[Van Schaik and Burkart 2011] Carel P Van Schaik and Judith M Burkart. Social learning and evolution: the cultural intelligence hypothesis. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 366(1567):1008–1016, 2011.

[Vinyals *et al.* 2017] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[Willi *et al.* 2022] Timon Willi, Alistair Hp Letcher, Johannes Treutlein, and Jakob Foerster. Cola: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, pages 23804–23831. PMLR, 2022.

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

[Williams 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

[Wolpert and Tumer 2001] David H Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(02n03):265–279, 2001.

[Wu *et al.* 2021] Zifan Wu, Chao Yu, Deheng Ye, Junge Zhang, Hankz Hankui Zhuo, et al. Coordinated proximal policy optimization. *Advances in Neural Information Processing Systems*, 34:26437–26448, 2021.

[Xing *et al.* 2021] Dong Xing, Qianhui Liu, Qian Zheng, Gang Pan, and ZH Zhou. Learning with generated teammates to achieve type-free ad-hoc teamwork. In *IJCAI*, pages 472–478, 2021.

[Yang *et al.* 2021] Tianpei Yang, Hongyao Tang, Chenjia Bai, Jinyi Liu, Jianye Hao, Zhaopeng Meng, and Peng Liu. Exploration in deep reinforcement learning: a comprehensive survey. *arXiv preprint arXiv:2109.06668*, 2021.

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

[Zhang and Lesser 2010] Chongjie Zhang and Victor Lesser. Multi-agent learning with policy prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 927–934, 2010.

[Zhang *et al.* 2021] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[Zhao *et al.* 2022] Stephen Zhao, Chris Lu, Roger Baker Grosse, and Jakob Nicolaus Foerster. Proximal learning with opponent-learning awareness. *arXiv preprint arXiv:2210.10125*, 2022.

[Zhou *et al.* 2020] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 33:11853–11864, 2020.

[Ziebart *et al.* 2008] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[Ziebart 2010] Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, USA, 2010.

Appendix

The following additional graphs, excluded from the main text of this dissertation, are presented to provide further insight into the empirical performance during training for each approach. Accompanying each figure is a caption detailing the specific experiment's results being depicted. Figure 1 displays the rewards of various approaches in the Simple Spread environment. In Figure 2, the rewards for each approach in the Space Invaders Environment are illustrated. Subsequently, Figure 3 presents the length of each episode in Space Invaders in terms of the number of steps taken. Figure 4 highlights the rewards for each approach in Cooperative Pong, while Figure 5 reveals the number of steps each approach required for each episode during training. Figures 6, 7 and 8 demonstrate the reward for each approach in our ablation study of the various cooperative environments. Lastly, Figures 9, 10, 11 and 12 demonstrate the performance of each approach in the Round-Robin in terms of rewards attained.

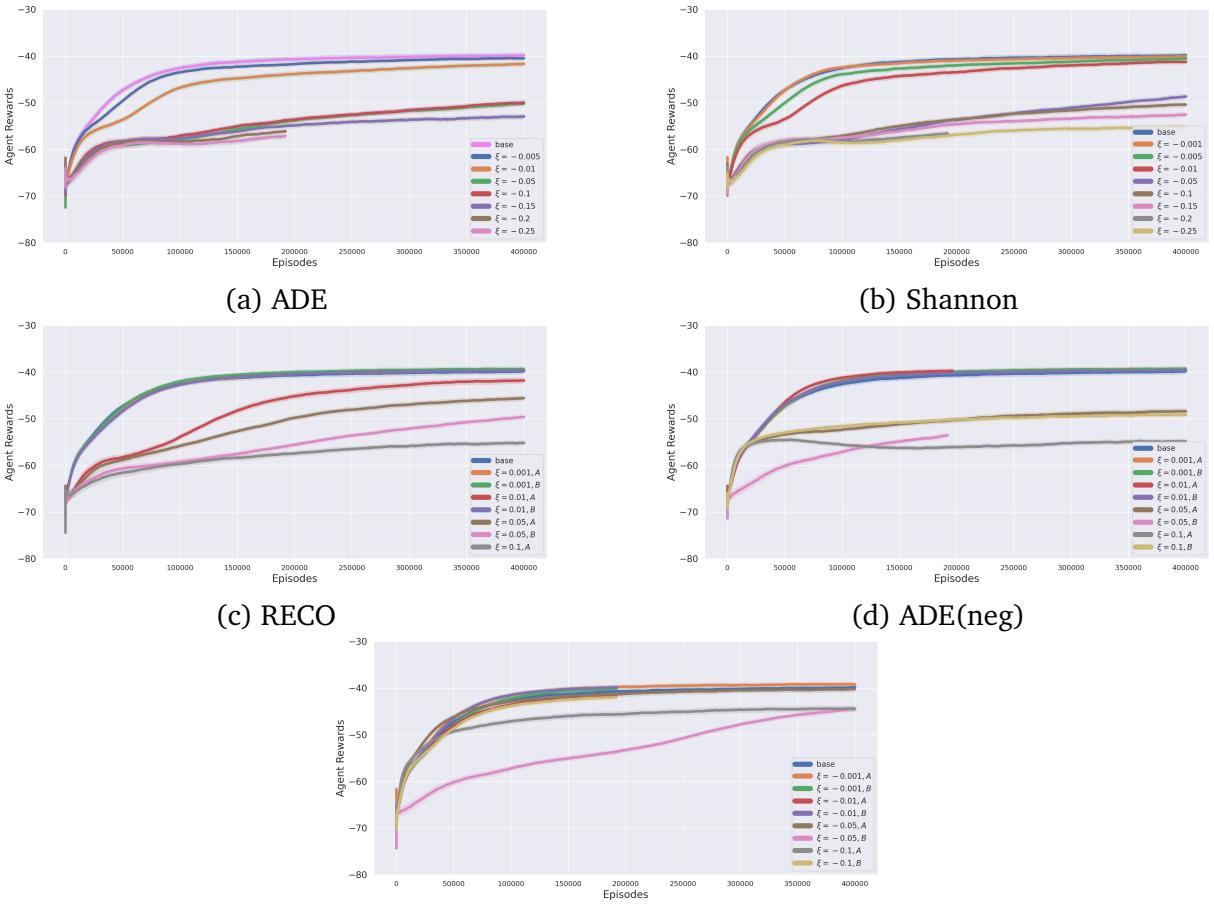
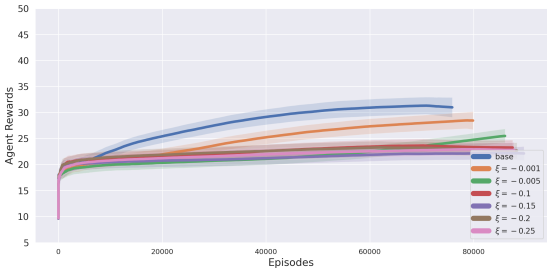
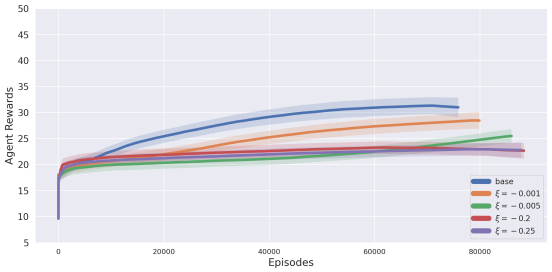


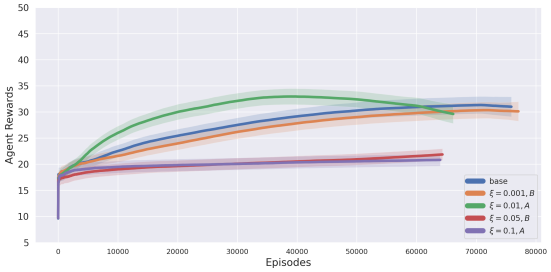
Figure 1: Learning curve showing the reward for episode for each approach from hyperparameter tuning in the Simple Spread environment (higher is better).



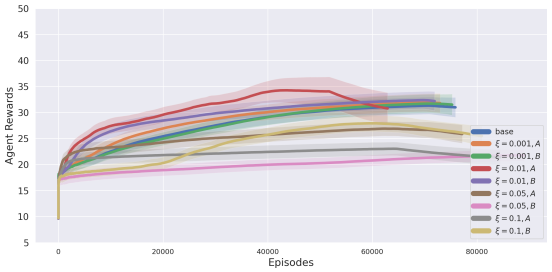
(a) ADE



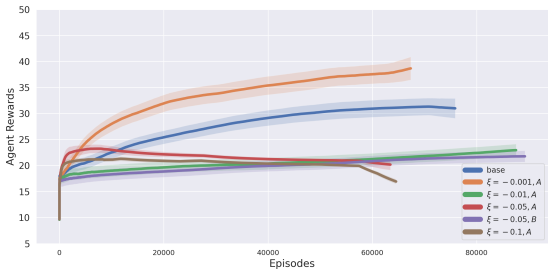
(b) Shannon



(c) RECO

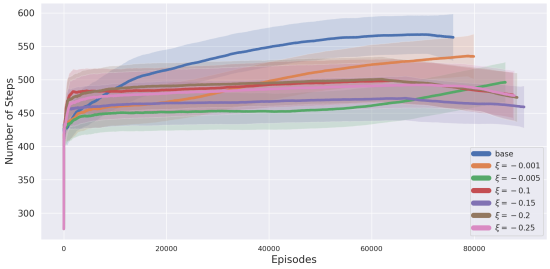


(d) ADE(neg)



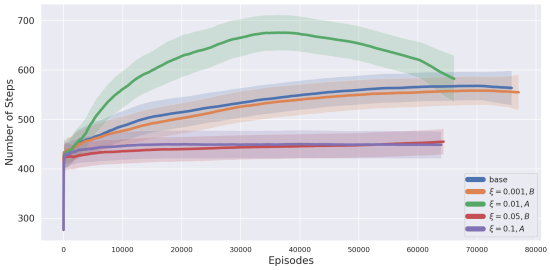
(e) RECO(neg)

Figure 2: Learning curve showing the reward for episode for each approach from hyperparameter tuning in the Space Invaders environment (higher is better).

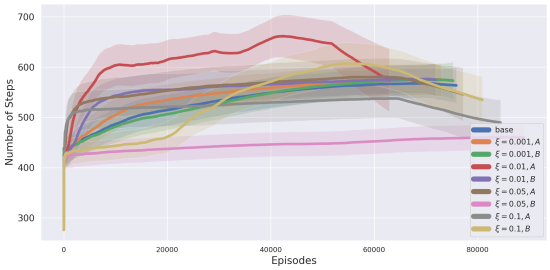


(a) ADE

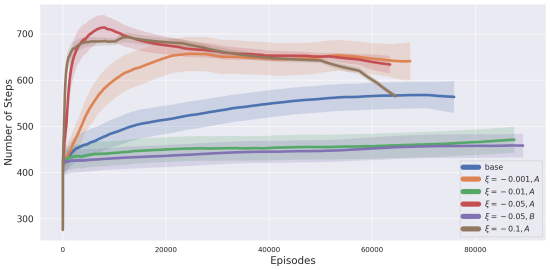
(b) Shannon



(c) RECO

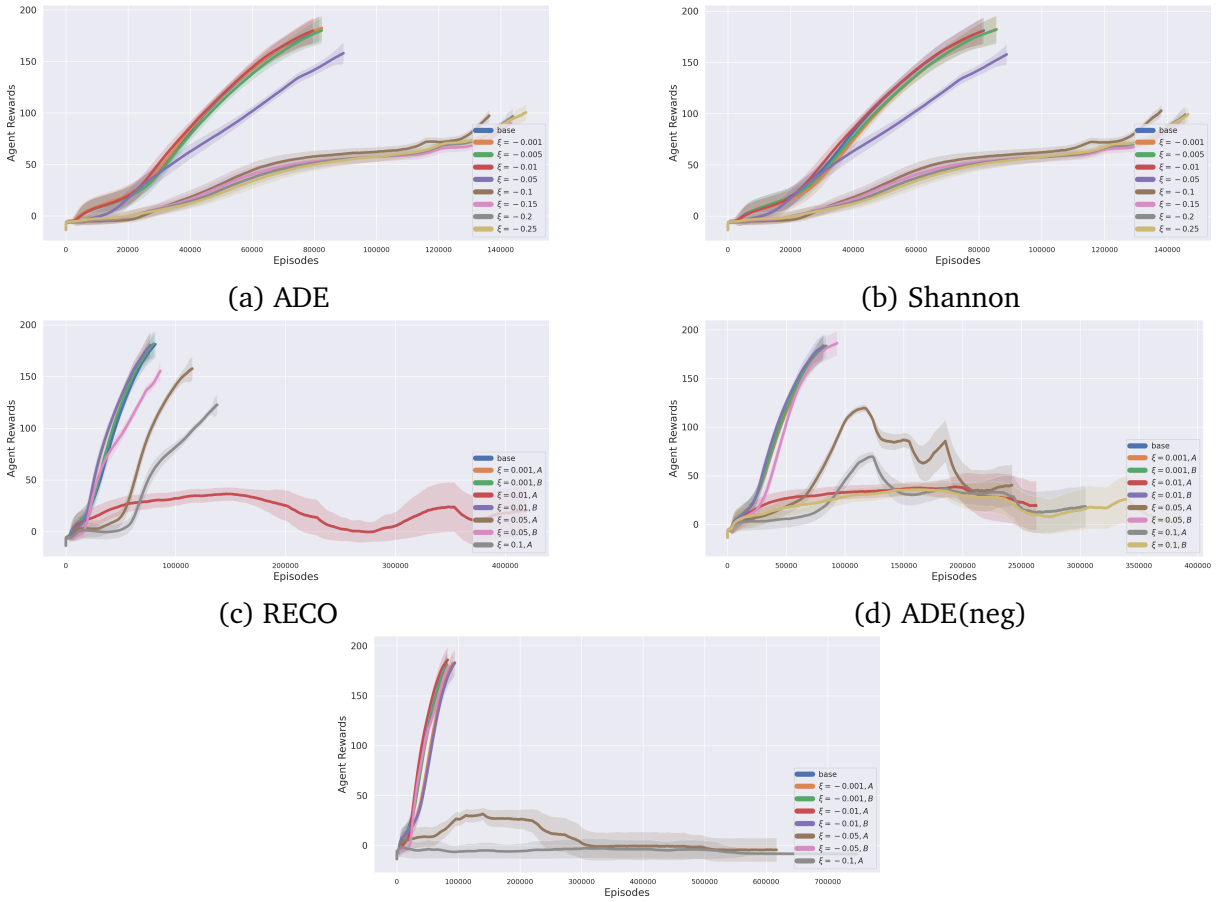


(d) ADE(neg)



(e) RECO(neg)

Figure 3: Learning curve showing the length of the episode (in terms of the number of steps) for each approach from hyperparameter tuning in the Space Invaders environment.



(a) ADE

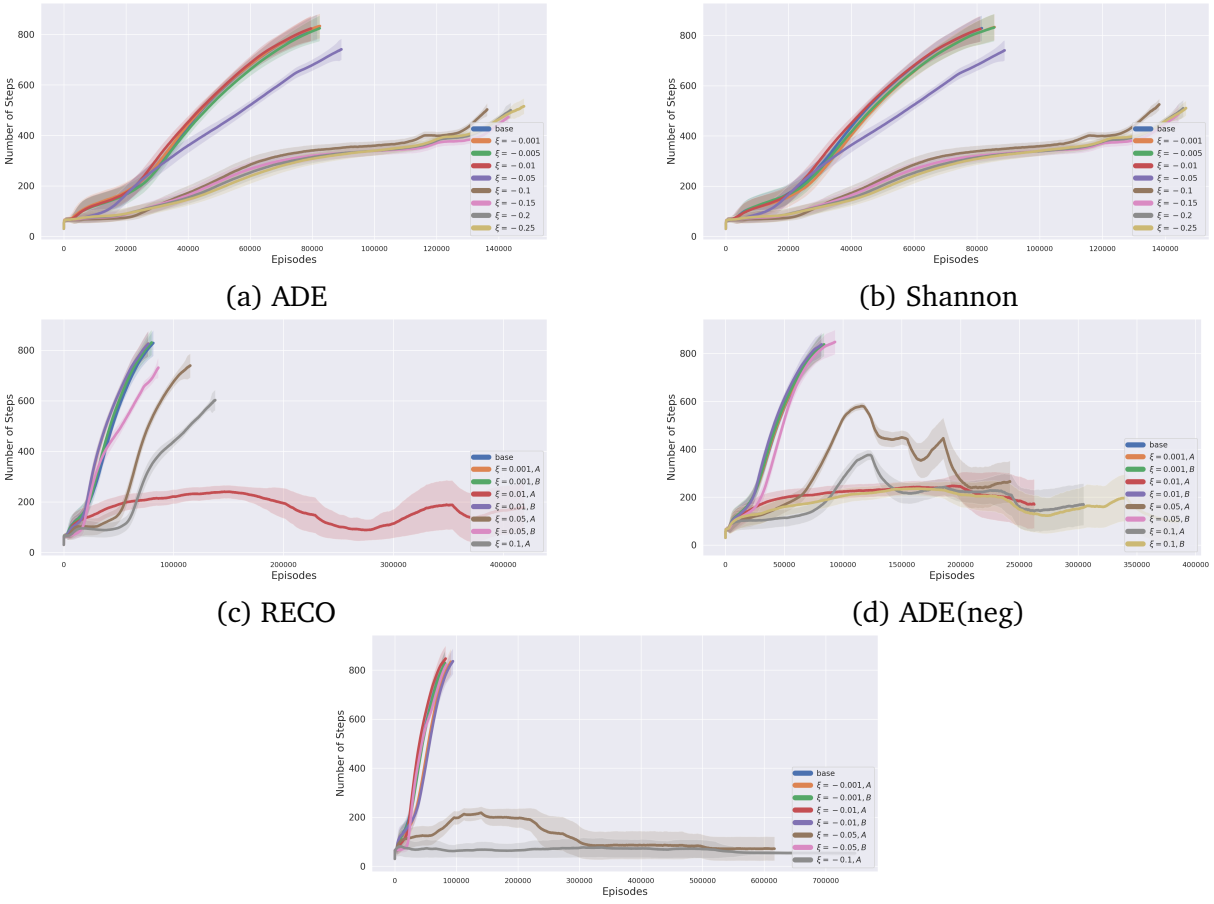
(b) Shannon

(c) RECO

(d) ADE(neg)

(e) RECO(neg)

Figure 4: Learning curve showing the reward for episode for each approach from hyperparameter tuning in the Cooperative Pong environment (higher is better).



(a) ADE

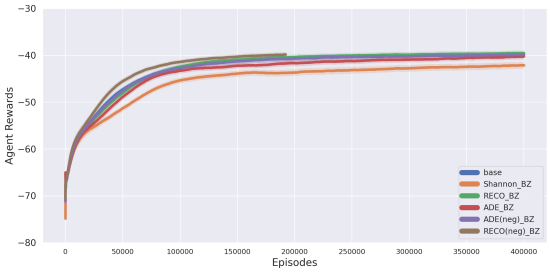
(b) Shannon

(c) RECO

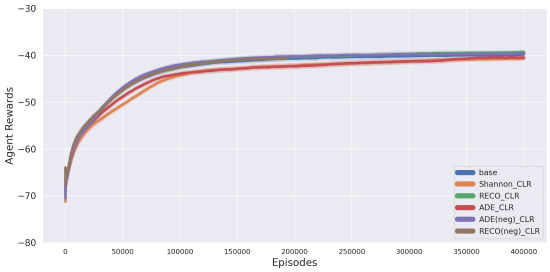
(d) ADE(neg)

(e) RECO(neg)

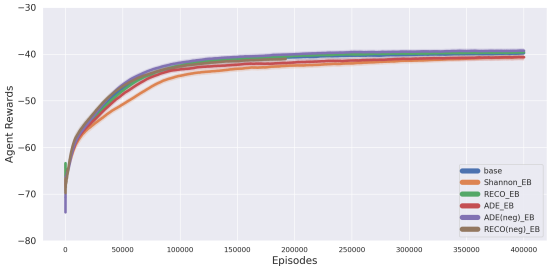
Figure 5: Learning curve showing the length of the episode (in terms of the number of steps) for each approach from hyperparameter tuning in the Cooperative Pong environment.



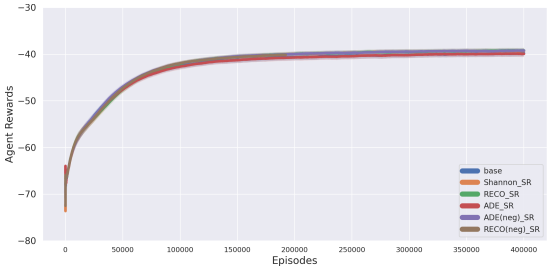
(a) Base Zeta



(b) Clear Entropy

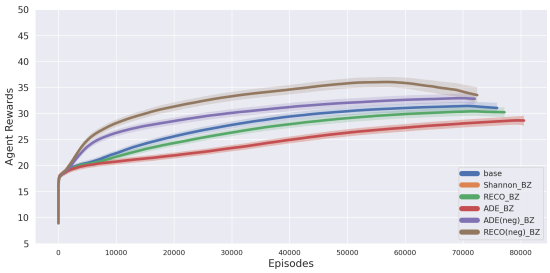


(c) Entropy Bonus

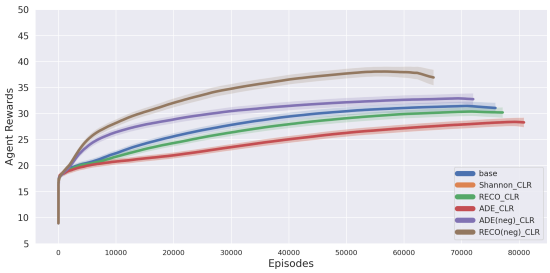


(d) Surrogate Function

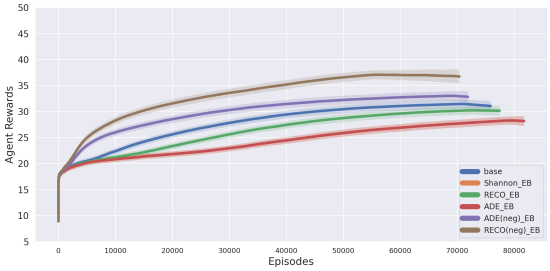
Figure 6: Learning curve showing the reward for episode for each approach from ablation in the Simple Spread environment (higher is better).



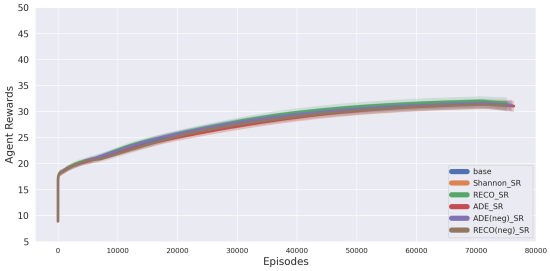
(a) Base Zeta



(b) Clear Entropy

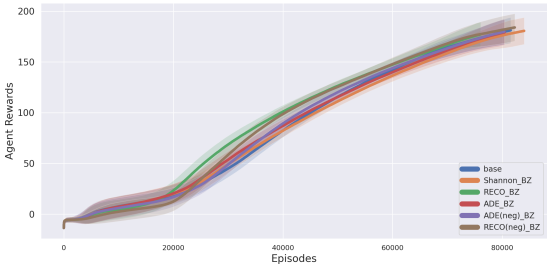


(c) Entropy Bonus

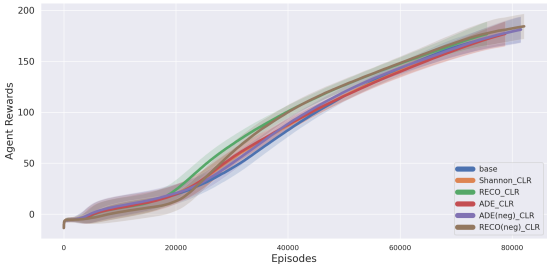


(d) Surrogate Function

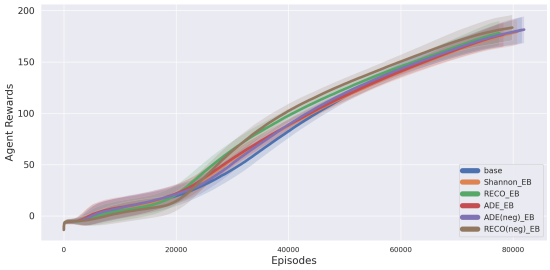
Figure 7: Learning curve showing the reward for episode for each approach from ablation in the Space Invaders environment (higher is better).



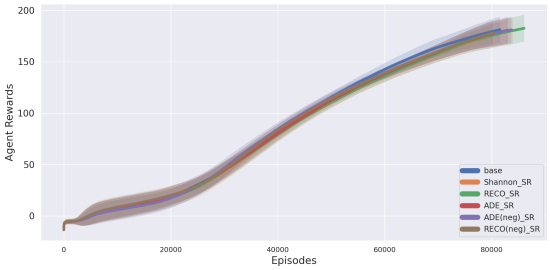
(a) Base Zeta



(b) Clear Entropy

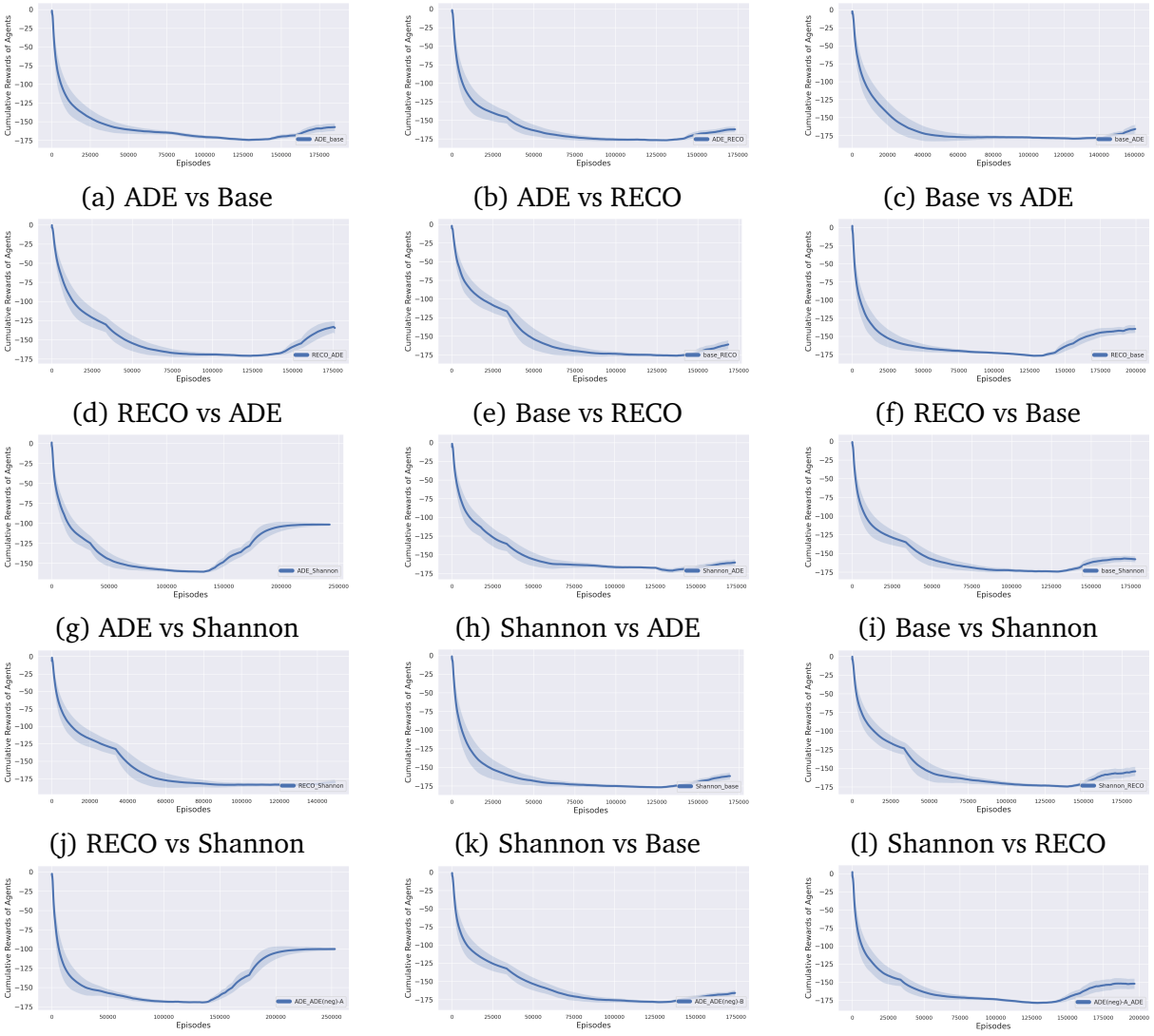


(c) Entropy Bonus



(d) Surrogate Function

Figure 8: Learning curve showing the reward for episode for each approach from ablation in the Cooperative Pong environment (higher is better).



(a) ADE vs Base (b) ADE vs RECO (c) Base vs ADE

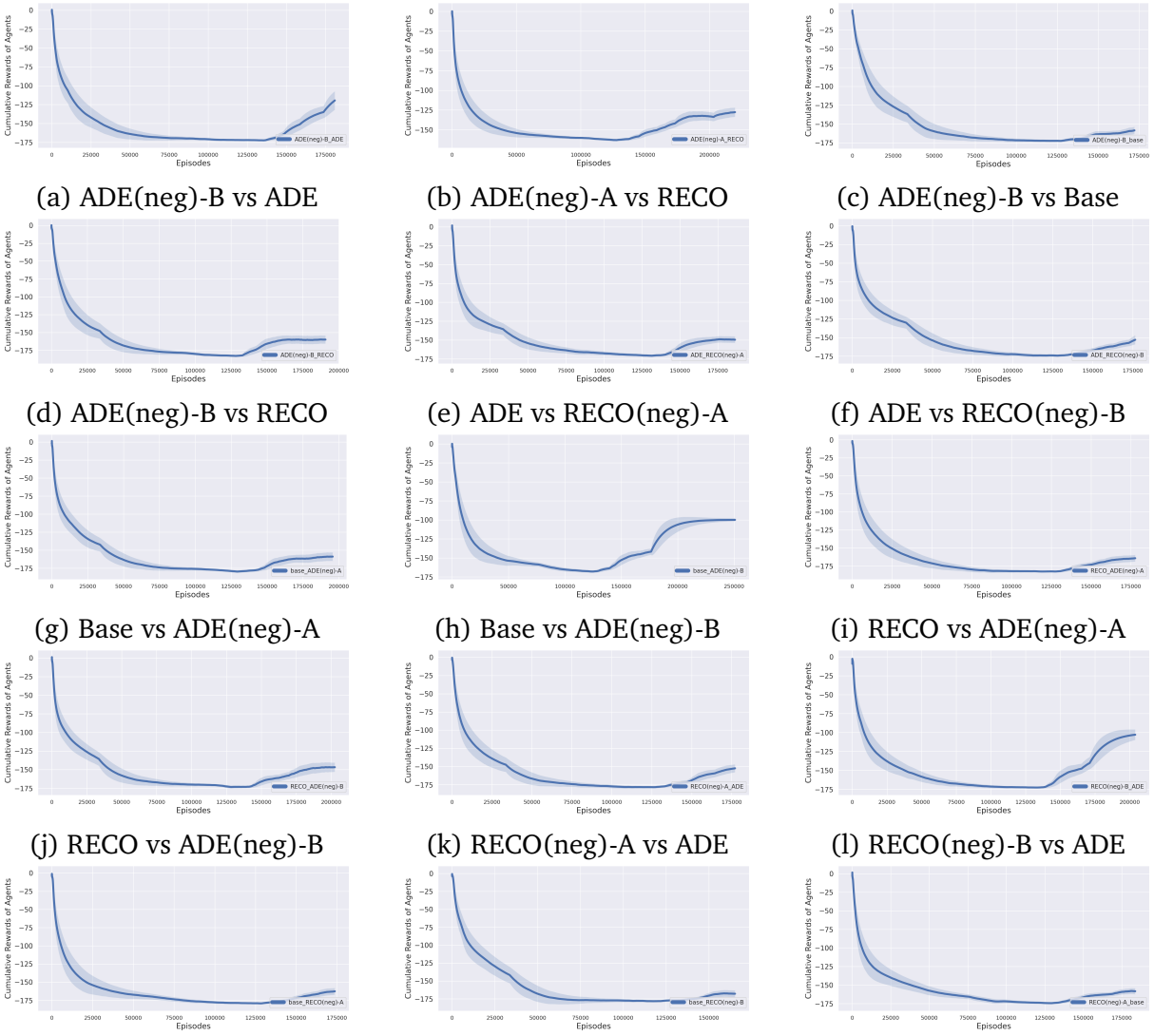
(d) RECO vs ADE (e) Base vs RECO (f) RECO vs Base

(g) ADE vs Shannon (h) Shannon vs ADE (i) Base vs Shannon

(j) RECO vs Shannon (k) Shannon vs Base (l) Shannon vs RECO

(m) ADE vs ADE(neg)-A (n) ADE vs ADE(neg)-B (o) ADE(neg)-A vs ADE

Figure 9: The positional bias in Boxing can be seen in these examples. Each subcaption indicates the competing approaches, with player one mentioned first, followed by player two. Here, we are representing rewards from player one's point of view hence, higher is better. These examples illustrate that in both games, player one is at a disadvantage to player two, regardless of the approach they use.



(a) ADE(neg)-B vs ADE (b) ADE(neg)-A vs RECO (c) ADE(neg)-B vs Base

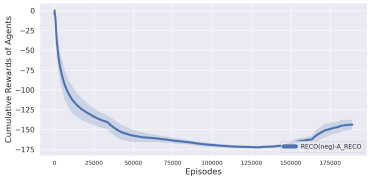
(d) ADE(neg)-B vs RECO (e) ADE vs RECO(neg)-A (f) ADE vs RECO(neg)-B

(g) Base vs ADE(neg)-A (h) Base vs ADE(neg)-B (i) RECO vs ADE(neg)-A

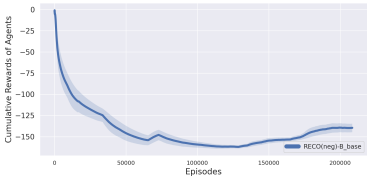
(j) RECO vs ADE(neg)-B (k) RECO(neg)-A vs ADE (l) RECO(neg)-B vs ADE

(m) Base vs RECO(neg)-A (n) Base vs RECO(neg)-B (o) RECO(neg)-A vs Base

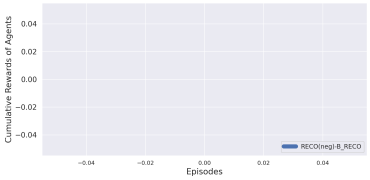
Figure 10: The positional bias in Boxing can be seen in these examples. Each subcaption indicates the competing approaches, with player one mentioned first, followed by player two. Here, we are representing rewards from player one's point of view hence, higher is better. These examples illustrate that in both games, player one is at a disadvantage to player two, regardless of the approach they use.



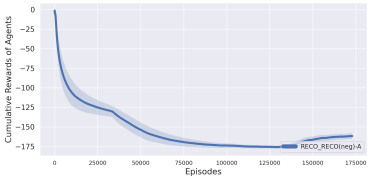
(a) RECO(neg)-A vs RECO



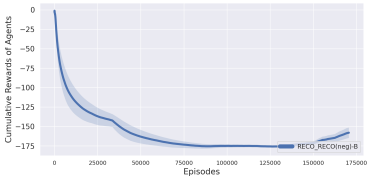
(b) RECO(neg)-B vs Base



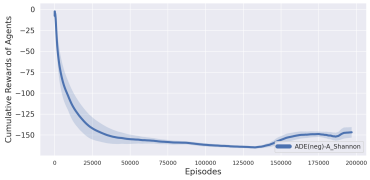
(c) RECO(neg)-B vs RECO



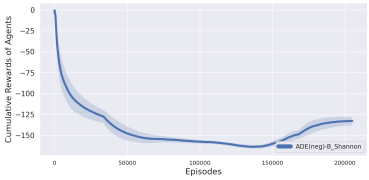
(d) RECO vs RECO(neg)-A



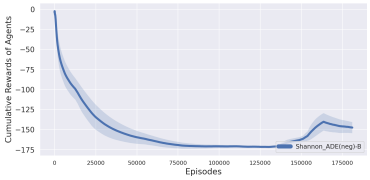
(e) RECO vs RECO(neg)-B



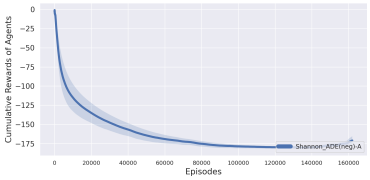
(f) ADE(neg)-A vs Shannon



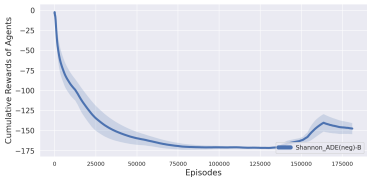
(g) ADE(neg)-B vs Shannon



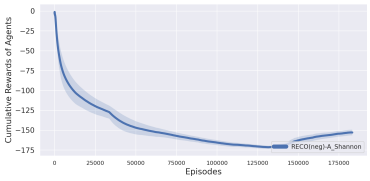
(h) Shannon vs ADE(neg)-B



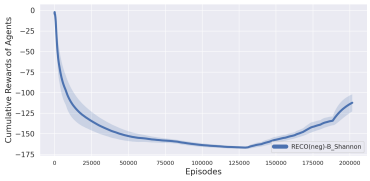
(i) Shannon vs ADE(neg)-A



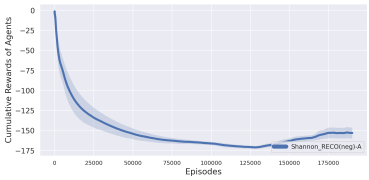
(j) Shannon vs ADE(neg)-B



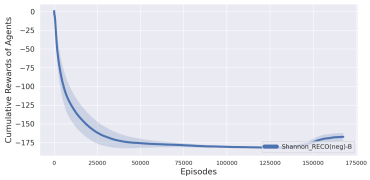
(k) RECO(neg)-A vs Shannon



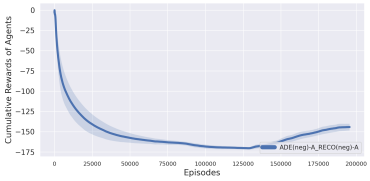
(l) RECO(neg)-B vs Shannon



(m) Shannon vs RECO(neg)-A

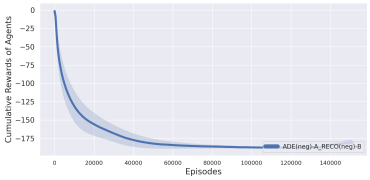


(n) Shannon vs RECO(neg)-B

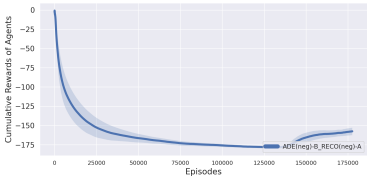


(o) ADE(neg)-A vs RECO(neg)-A

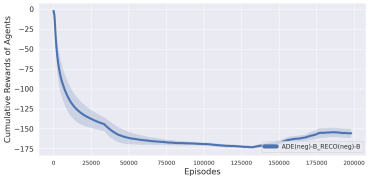
Figure 11: The positional bias in Boxing can be seen in these examples. Each subcaption indicates the competing approaches, with player one mentioned first, followed by player two. Here, we are representing rewards from player one's point of view hence, higher is better. These examples illustrate that in both games, player one is at a disadvantage to player two, regardless of the approach they use.



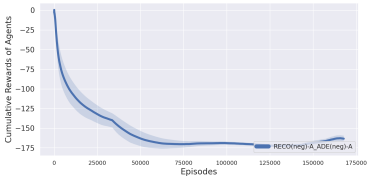
(a) ADE(neg)-A vs RECO(neg)-B



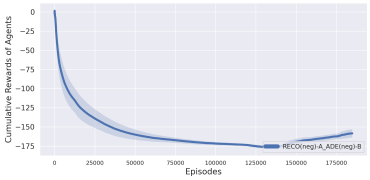
(b) ADE(neg)-B vs RECO(neg)-A



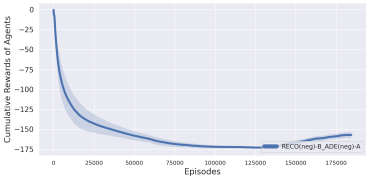
(c) ADE(neg)-B vs RECO(neg)-B



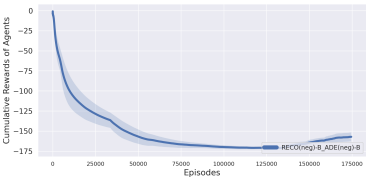
(d) RECO(neg)-A vs ADE(neg)-A



(e) RECO(neg)-A vs ADE(neg)-B



(f) RECO(neg)-B vs ADE(neg)-A



(g) RECO(neg)-B vs ADE(neg)-B

Figure 12: The positional bias in Boxing can be seen in these examples. Each subcaption indicates the competing approaches, with player one mentioned first, followed by player two. Here, we are representing rewards from player one's point of view hence, higher is better. These examples illustrate that in both games, player one is at a disadvantage to player two, regardless of the approach they use.