Understanding Learning in Multi-Agent Reinforcement Learning

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

Kinal Mehta
2020701019
kinal.mehta@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500032, INDIA
October 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " **Understanding Learning in Multi-Agent Reinforcement Learning** " by **Kinal Mehta**, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Pawan Kumar

To *Prosperity*

Acknowledgments

I would like to express my heartfelt gratitude to my parents and brother for their unwavering support, encouragement, and care throughout my academic journey. Their emotional support during low phases was invaluable to me, and I cannot thank them enough. I would also like to give a special shoutout to my brother Fenil, who also helped me with numerous debugging sessions. I am truly blessed to have such a loving family.

I am deeply indebted to my advisor, Prof. Pawan Kumar, for presenting me with the opportunity to explore my interests in Reinforcement Learning and guiding me throughout the process. His expertise, patience, and support were instrumental in shaping my research and helping me overcome obstacles. I would also like to express my gratitude to my collaborator, Anuj, for his unwavering support and being my go-to person for any RL discussion. His insightful comments and suggestions have been invaluable to me and have helped me immensely in refining my research. I feel truly fortunate to have worked alongside him.

I would like to extend my appreciation to Prof. Madhav Krishna for his support and guidance during my time at the Robotics Research Center. His expertise, mentorship, and encouragement have been invaluable in shaping my research and professional development.

I would like to express my heartfelt gratitude to my friends at the university, who have been a constant source of support and inspiration. I am grateful to Harshit, Shantanu, Abhinaba, Udit, Aniket, Neel, Amit, Krishna, Swayatta, Sanjay, Pramod, Ritam, and many others whose names are not listed here, for their encouragement, stimulating discussions, and their friendship throughout my academic journey. I will always cherish our friendship and the joyous moments that we shared.

I am deeply grateful to all those who have contributed to my academic and personal development. Their support and encouragement have been invaluable, and I will always cherish their kindness and generosity.

Abstract

Reinforcement Learning (RL) has witnessed remarkable advancements in both algorithms and engineering, enabling a wide range of exciting applications. Multi-agent Reinforcement Learning (MARL) in particular has made strides of progress enabling multiple learning entities to interact in effective manner. Few of the challenges that still remain are learning under sparse rewards, achieving social generalization by adapting to changing behaviours of other agents and reproducibility. This thesis tackles two important challenges in MARL: (1) learning in multi-agent sparse reward environments and (2) reproducibility with social generalization.

In the first part, we address the issue of learning a reliable critic in multi-agent sparse reward scenarios. The exponential growth of the joint action space with the number of agents, coupled with reward sparsity and environmental noise, poses significant hurdles for accurate learning. To mitigate these challenges, we propose regularizing the critic with spectral normalization (SN). Our experiments demonstrate that the regularized critic exhibits improved robustness, enabling faster learning even in complex multi-agent scenarios. These findings highlight the importance of critic regularization for stable learning.

In the second part, we introduce marl-jax, a powerful software package for MARL that focuses on training and evaluating social generalization of agents. Built on DeepMind's JAX ecosystem and leveraging their RL framework, marl-jax supports cooperative and competitive environments with multiple agents acting simultaneously. It provides an intuitive command-line interface for training agent populations and evaluating their generalization capabilities. Researchers interested in exploring social generalization in MARL can leverage marl-jax as a reliable baseline.

In conclusion, this thesis addresses two crucial challenges in RL: learning in multi-agent sparse reward scenarios and reproducibility for social generalization in MARL. By introducing spectral normalization as a regularization technique and providing the marl-jax software package, this research contributes to enhancing stability, robustness, social generalization and reproducibility in RL.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study of how humans learn in the presence of multiple learning agents, including other humans, is closely related to Multi-Agent Reinforcement Learning (MARL). It explores how individuals acquire knowledge, skills, and behaviors in social contexts where multiple agents, including humans, are simultaneously learning and adapting their behaviors.

Humans are highly adaptive and capable of learning from a variety of sources, including interactions with other individuals. In the presence of multiple learning agents, humans engage in a complex process of social learning, where they observe, imitate, and communicate with others to acquire new knowledge and skills. This social learning process is influenced by various factors, such as social norms, cultural influences, social hierarchies, and individual differences.

Examining how humans learn in social contexts provides researchers with a rich source of knowledge about effective learning strategies, communication methods, and coordination mechanisms. By observing how humans acquire knowledge and skills from others, MARL researchers can identify key factors that contribute to successful learning in multi-agent environments. These insights can then be integrated into MARL algorithms to improve learning efficiency, coordination, and cooperation among agents.

Additionally, understanding how humans adapt their strategies and behaviors in response to other agents' actions can inform the development of MARL algorithms that promote adaptive and flexible decision-making. Humans possess the ability to dynamically adjust their behaviors based on the observed behavior of other agents, and MARL algorithms can adopt similar adaptive mechanisms to enhance their performance in complex multi-agent environments.

1.1 Contributions

In this thesis we address two important challenges in Multi-agent Reinforcement Learning

1. Sparse reward learning in Cooperative MARL by regularizing the critic using Spectral Normalization

- We introduce a sparse reward configuration for SMAC and show that it is difficult to learn when compared to standard reward configuration
- We propose to regularise the critic with spectral normalization and show that it helps learn better policies under sparse rewards
- We analyse the effects of applying spectral normalization and show that it helps 1) stabilise the critic gradients and 2) has an optimization effect of scaling the gradients of the entire critic by the product of the largest spectral value of the weight matrices

2. Reproducibility in MARL by developing marl-jax, a scalable framework for social generalization in MARL
    - We develop marl-jax, a JAX based framework for training and evaluating social generalization of trained MARL policies
    - The framework is loaded with features such as multi-GPU training and distributed experience collection enabling large scale experimentation
    - The proposed framework is evaluated on two commonly used benchmarks Meltingpot [1] and Overcooked [7]

1.2 Thesis Organization

This thesis is divided into six parts and is organised as follows

- First chapter motivates the problem of multi-agent reinforcement learning and the contributions of this work in addressing the two important challenges in MARL
- Second chapter provides with relevant background on reinforcement learning, multi-agent reinforcement learning, MDP formulation and its extensions under partial observability and multi-agent environments and spectral normalization and its impact on deep learning
- Third chapter talks about the problem of learning in sparse rewards cooperative multi-agent learning and proposes a regularization based solution to address the problem
- Fourth chapter describes about the *marl-jax* framework and its various implementation details with the benchmark results
- Fifth chapter concludes the thesis and its contributions and motivates further plausible directions of research in multi-agent learning and social generalization
- Sixth chapter mentions the resulting publications from this work

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning that focuses on training agents to make decisions in an environment to maximize a cumulative reward signal. RL is modeled as a Markov Decision Process (MDP), which consists of a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition probability function, $\mathcal{R}$ is the reward function, and γ is the discount factor [52].

The goal of RL is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions that maximize the expected cumulative reward, also known as the return. The return G_t at time t is defined as the sum of discounted rewards from time t to the end of the episode:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} R_k, \tag{2.1}$$

where T is the time step when the episode ends, R_k is the reward at time k, and γ is the discount factor that controls the importance of future rewards.

RL algorithms can be categorized into value-based methods, policy-based methods, and actor-critic methods [30]. Value-based methods learn a value function that estimates the expected return from a state or state-action pair. The value function $V^\pi(s)$ for a policy π is defined as the expected return starting from state s:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]. \tag{2.2}$$

Policy-based methods learn a policy directly by optimizing a parameterized policy function $\pi_\theta(u \mid s)$ that maps states to action probabilities. The policy gradient $\nabla_\theta J(\theta)$ is defined as the gradient of the expected return with respect to the policy parameters:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(u \mid s) Q^\pi(s, u) \right], \tag{2.3}$$

where $Q^\pi(s, u)$ is the state-action value function that estimates the expected return starting from state s, taking action u, and following policy π thereafter.

Actor-critic methods combine both value-based and policy-based methods by learning an actor that selects actions and a critic that estimates the value function [24]. The actor's parameters are updated using the policy gradient, while the critic's parameters are updated using temporal difference (TD) learning [51]. The TD error δ_t at time t is defined as the difference between the observed reward and the estimated value:

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2.4}$$

where s_t and s_{t+1} are the current and next states, respectively.

In conclusion, RL is a powerful framework for modeling decision-making problems, and various algorithms have been proposed to learn policies that maximize the expected return, including value-based methods

2.1.1 Markov Decision Process (MDP)

$$\text{MDP:} \quad \{S, U, P, r, \gamma\},$$

where:

- S represents the set of states in the MDP
- U represents the set of actions or control inputs available to the agent
- P is the state transition probability matrix, where $P(s, u, s')$ represents the probability of transitioning from state s to state s' under action u
- r represents the reward function, where $r(s, u)$ denotes the immediate reward received when taking action u in state s
- γ is the discount factor, determining the trade-off between immediate and future rewards

The MDP formulation allows us to mathematically model the interaction between an agent and its environment in a sequential decision-making setting. The agent's goal is to learn a policy $\pi(u|s)$ that maximizes the expected cumulative reward over time. The policy specifies the action to be taken in each state, and the agent aims to find the optimal policy π^* that maximizes the expected cumulative reward.

The dynamics of the MDP can be expressed using the Bellman equation, which represents the value function $V^\pi(s)$ for a given policy π:

$$V^\pi(s) = \sum_{u \in U} \pi(u|s) \left(r(s, u) + \gamma \sum_{s' \in S} P(s, u, s') V^\pi(s') \right).$$

The optimal value function $V^*(s)$ can be defined as the maximum value over all policies:

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

Similarly, the optimal action-value function $Q^*(s, u)$ represents the maximum expected cumulative reward when taking action u in state s:

$$Q^*(s, u) = \sum_{s' \in S} P(s, u, s') \left(r(s, u) + \gamma \max_{u' \in U} Q^*(s', u') \right).$$

Solving the MDP involves finding the optimal policy π^* or the optimal action-value function Q^* through methods such as value iteration, policy iteration, or reinforcement learning algorithms.

2.1.2 Partially Observable Markov Decision Process (POMDP)

$$\text{POMDP:} \quad \{S, U, P, O, r, \gamma\}$$

- S represents the set of states in the POMDP
- U represents the set of actions or control inputs available to the agent
- P is the state transition probability matrix, where $P(s, u, s')$ represents the probability of transitioning from state s to state s' under action u
- O is the observation probability matrix, where $O(o|s, u)$ represents the probability of observing observation o given the agent is in state s and takes action u
- r represents the reward function, where $r(s, u)$ denotes the immediate reward received when taking action u in state s
- γ is the discount factor, determining the trade-off between immediate and future rewards

In a POMDP, the agent does not have direct access to the underlying states but receives observations based on the true state. The agent maintains a belief state b that represents its current belief or probability distribution over the underlying states. The belief state is updated using the observed information and the transition and observation probabilities.

The agent's goal in a POMDP is to learn a policy $\pi(u|b)$ that maximizes the expected cumulative reward over time. The policy specifies the action to be taken based on the current belief state, and the agent aims to find the optimal policy π^* that maximizes the expected cumulative reward.

2.1.3 Policy Gradient

Policy gradient is an algorithm used in Reinforcement Learning (RL) to learn policies directly without explicitly estimating value functions. It is especially effective in situations where the action space is large or continuous.

The main idea behind policy gradient is to optimize the parameters of a policy directly by estimating the gradient of a performance measure with respect to these parameters. The performance measure typically represents the expected cumulative reward obtained by following the policy.

Mathematically, let θ denote the parameters of the policy π_θ. The policy gradient can be computed as:

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta E_{\tau\sim\pi_\theta}[R(\tau)] \tag{2.5}$$

$$= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \tag{2.6}$$

$$= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \tag{2.7}$$

$$= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) \tag{2.8}$$

$$= E_{\tau\sim\pi_\theta}\left[\nabla_\theta \log P(\tau|\theta)R(\tau)\right], \tag{2.9}$$

where, $R(\tau) = G_0$, that is the return of the episode starting at timestep zero, or the return of the entire episode.

Probability of a Trajectory The probability of a trajectory $\tau = (s_0, a_0, ..., s_{T+1})$ given that actions come from π_θ can be written as

$$P(\tau|\theta) = \rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, u_t)\pi_\theta(u_t|s_t). \tag{2.10}$$

Log-Probability of a Trajectory The log-prob of a trajectory by taking log on both sides of the above equation is

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T}\Big(\log P(s_{t+1}|s_t, u_t) + \log \pi_\theta(u_t|s_t)\Big). \tag{2.11}$$

Taking gradient of Log-Probability of Trajectory with respect to policy parameters

$$\nabla_\theta \log P(\tau|\theta) = \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T}\Big(\cancel{\nabla_\theta \log P(s_{t+1}|s_t, u_t)} + \nabla_\theta \log \pi_\theta(u_t|s_t)\Big) \tag{2.12}$$

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(u_t|s_t). \tag{2.13}$$

Final Policy Gradient Equation

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(u_t|s_t)R(\tau)\right]. \tag{2.14}$$

2.1.3.1 Actor-Critic Methods

Actor-Critic methods algorithms combine the advantages of both policy-based and value-based approaches. These methods extend the basic policy gradient algorithm by incorporating a value function estimation component known as the critic.

In traditional policy gradient methods, the agent learns a policy directly and updates its parameters based on the expected cumulative reward. However, these methods can suffer from high variance and slow convergence since they rely solely on the reward signal. Value-based methods, on the other hand, estimate the value function to guide the learning process, but they may struggle with exploration and suffer from bias in the value estimation.

Actor-Critic methods bridge the gap between policy-based and value-based methods by introducing a critic network that estimates the value function. The critic provides an estimate of the expected cumulative reward, which is used to evaluate the quality of actions taken by the policy. The actor component updates the policy parameters based on the advantage or the difference between the estimated value and the actual observed rewards.

The interaction between the actor and the critic leads to a more stable and efficient learning process. The critic helps to reduce the variance in the policy updates by providing a baseline for comparison, while the actor focuses on improving the policy based on the advantage information.

Actor-Critic methods have shown significant success in various RL tasks, including both single-agent and multi-agent scenarios. They offer a balance between exploration and exploitation, provide stability in learning.

$$\text{Objective} = \mathbb{E}_t \left[\log \pi_\theta(u_t|s_t) \cdot A_t(s_t, u_t; \phi) \right] \tag{2.15}$$

In this equation, $\pi_\theta(u_t|s_t)$ represents the actor's policy, which specifies the probability of taking action u_t in state s_t given the actor's parameters θ. $A_t(s_t, u_t; \phi)$ denotes the advantage function, which measures the advantage or value of taking action u_t in state s_t based on the critic's parameters ϕ. The advantage function compares the value of an action with the expected cumulative reward under the current policy.

2.1.3.2 Advantage Function

The advantage function measures the advantage of taking a specific action in a given state compared to the expected cumulative reward under the current policy. It is defined as the difference between the estimated state-action value function and the estimated state value function:

$$A(s_t, u_t) = G_t - V_t(s_t) \tag{2.16}$$

$$A(s_t, u_t) = Q_t(s_t, u_t) - V_t(s_t). \tag{2.17}$$

We can also write the advantage function as $A_t = A(s_t, u_t)$.

2.1.3.3 Proximal Policy Optimization

PPO is based on the actor-critic method and employs a clipped surrogate objective function to ensure stable updates. The algorithm iteratively collects trajectories by interacting with the environment and uses these trajectories to update the policy parameters.

The key idea behind PPO is to strike a balance between exploration and exploitation while updating the policy. By limiting the policy updates, PPO maintains a certain level of exploration, allowing the algorithm to explore new and potentially better actions. At the same time, it also exploits the information learned from the previous policy, avoiding drastic changes that could lead to instability.

It achieves this by using a surrogate objective function that consists of two components: a policy ratio term and a clipping term. The policy ratio term measures the advantage of the new policy compared to the old policy, while the clipping term constrains the policy updates to be within a specified range.

Mathematically, the objective function of PPO can be defined as:

$$L(\theta) = \mathbb{E}t \left[\min \left(\frac{\pi_\theta(u_t|s_t)}{\pi_{\theta_{\text{old}}}(u_t|s_t)} A_t, \text{clip} \left(\frac{\pi_\theta(u_t|s_t)}{\pi_{\theta_{\text{old}}}(u_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right], \tag{2.18}$$

where θ and θ_{old} represent the current and previous policy parameters, $\pi_\theta(u_t|s_t)$ is the probability of taking action u_t in state s_t under the current policy, A_t is the advantage function that measures the advantage of taking action u_t in state s_t, and ϵ is a hyperparameter that controls the extent of the clipping.

The objective function is maximized through stochastic gradient ascent, where the policy parameters are updated using gradient-based optimization techniques such as Adam or RMSprop.

2.2 Multi-agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is a subfield of reinforcement learning that focuses on developing algorithms and techniques for training multiple agents to interact and collaborate in an environment. In MARL, the agents can be autonomous entities with their own objectives, observations, and actions. The goal is to enable the agents to learn effective policies and behaviors through interaction with the environment and other agents.

Unlike single-agent reinforcement learning, where a single agent learns to optimize its behavior in isolation, MARL involves multiple agents that must take into account the actions and behaviors of other agents. The agents can have different levels of cooperation, from fully cooperative settings where all agents work together towards a common goal, to competitive settings where agents compete against each other, or mixed settings with a combination of cooperation and competition.

MARL presents unique challenges compared to single-agent RL. The presence of multiple agents introduces the issue of non-stationarity, as the policies of other agents can change during training. The agents must learn to adapt and respond to the evolving strategies of their counterparts. The state space

and action space also become more complex, as the agents' actions and observations are influenced by the actions and behaviors of other agents.

MARL has a wide range of applications, such as multi-robot systems, self-driving cars, traffic control, and multi-agent games. It enables the development of intelligent systems that can collaborate, compete, and interact in complex and dynamic environments.

2.2.1 Types of Agents

2.2.1.1 Homogeneous Agents

Homogeneous agents in Multi-Agent Reinforcement Learning (MARL) environments refer to scenarios where all the agents in the system have the same capabilities, goals, and learning algorithms. They have the same action space and state space. Homogeneous MARL environments often involve cooperative or competitive settings, where agents collaborate or compete with each other towards a common or individual objective.

In homogeneous MARL, agents learn and interact with the environment simultaneously, exchanging information and adapting their policies based on their observations and interactions. The agents can share knowledge, coordinate their actions, and collectively improve their performance over time.

2.2.1.2 Heterogeneous Agents

Heterogeneous agents in MARL environments refer to scenarios where agents have distinct characteristics, capabilities, goals, or learning algorithms. Each agent may have a different action space, observe different state information and have different state space, or pursue different objectives. Heterogeneous MARL environments often represent more complex and realistic settings, where agents possess diverse skills, strategies, or roles.

MARL environments with heterogeneous agents introduce additional challenges and opportunities. Agents may have different levels of expertise, specialization, or responsibilities, leading to complementary or conflicting objectives. Coordination and cooperation among heterogeneous agents become more intricate, as they need to adapt to the varying capabilities and behaviors of others.

2.2.2 Challenges in MARL

2.2.2.1 Non-Stationarity

Non-stationarity refers to the dynamic nature of the environment caused by the interaction of multiple agents. As agents learn and update their policies, the environment changes, making it difficult to maintain stable learning. The actions of other agents can be unpredictable, and the system's overall behavior can exhibit non-stationary patterns. Dealing with non-stationarity in MARL requires developing

algorithms and techniques that can adapt to changing environments and accommodate the interactions and learning of other agents.

Both IPPO and IQL face the challenge of non-stationarity in cooperative environments. In independent learning algorithms like IQL [53] and IPPO, the interactions between agents and the learning process can lead to changes in the environment dynamics and the behaviors of other agents. These changes result in non-stationary conditions, where the underlying assumptions of stationary environments for the learning algorithms no longer hold. To cope with this issue, approaches such as using a centralised critic, as seen in MADDPG [32], are employed to address the non-stationarity problem and stabilize the learning process in cooperative multi-agent scenarios.

In addition to approaches like MADDPG, other solutions have been proposed to address the non-stationarity challenge in cooperative multi-agent environments. Two notable examples are VDN (Value-Decomposition Networks) and QMIX (Q-Mix).

VDN [50] decomposes the global Q-function into individual Q-values for each agent, allowing each agent to learn its own value function while considering the joint action-value. By combining these individual Q-values, VDN ensures coordination and cooperation among the agents.

QMIX [46] takes a different approach by using a mixing network to combine individual agent's Q-values in a way that guarantees monotonicity in the joint action-value function. This ensures that the value function increases as the agents improve their policies, facilitating cooperative behavior.

Another approach that combines centralised training with decentralised execution is MAPPO (Multi-Agent Proximal Policy Optimization) [57]. MAPPO leverages a centralised critic during training to address non-stationarity and improve the learning stability, while allowing for decentralised execution during policy execution for better scalability and real-world applicability.

These solutions, including VDN, QMIX, and MAPPO, offer different strategies to handle the non-stationarity challenge in cooperative multi-agent learning. By incorporating centralized training, decentralised execution, value decomposition, and joint action-value mixing, these methods aim to improve the coordination and cooperation among agents and enhance the overall performance in cooperative multi-agent environments.

2.2.2.2 Generalization to Co-players

Generalization refers to the ability of agents to transfer their learned knowledge and skills to new scenarios or co-players. In MARL, agents may encounter novel situations or interact with previously unseen agents. Generalizing to co-players involves adapting and adjusting strategies to effectively collaborate or compete with unfamiliar agents. Generalization challenges arise due to the diversity of co-players and the need to adapt policies based on their behaviors. Ensuring that MARL algorithms can generalize to co-players is essential for robust and scalable multi-agent learning.

The "Options as Responses" (OPRE) [55] framework presents a hierarchical agent that exhibits the capability to generalize its behavior to opponents it has not encountered during training. The hierarchical architecture of OPRE is rooted in the game-theoretical structure of the environment. At the top level, the

agent selects strategic responses to opponents, while at the lower level, these responses are translated into a policy over low-level actions. By employing a hierarchical approach, OPRE enables the agent to effectively reason and adapt its behavior in response to different opponents. The top-level decision-making allows the agent to consider the overall strategic context of the game and choose appropriate responses. The low-level policy implementation ensures that these strategic responses are translated into specific actions in a fine-grained manner.

Another approach to address the challenge of diverse behavioral responses in multi-agent reinforcement learning is through a two-phase training process. In the first phase, a population of policies is trained to exhibit diverse behaviors. Then, in the second phase, this population of diverse policies is leveraged to train a Best Response policy that can generalize to the entire population.

TrajDi [33] introduces a training objective based on Jensen-Shannon divergence to encourage the generation of a diverse population of policies. By optimizing this objective, TrajDi promotes the exploration of different behavioral strategies among the agents, leading to a more varied population.

LIPO [10] employs a policy incompatibility objective to train a diverse population of policies. This objective encourages the policies to have different behavioral patterns that are not compatible with each other, resulting in a more diverse set of strategies among the agents.

Ranked Policy Memory (RPM) [43] presents a unique method inspired by the concept of a replay buffer. It maintains a buffer of policies and ranks/classifies them based on the episodic return obtained. By using this ranked policy memory, RPM trains a Best Response Policy that has the ability to generalize to co-player policies exhibiting a wide range of behaviors.

These approaches aim to promote diversity among the policies in multi-agent settings, allowing for a richer and more varied interaction between agents. By training a diverse population of policies or utilizing a ranked policy memory, these methods enhance the ability to generalize and adapt to different behavioral responses exhibited by other agents.

For a comprehensive review of generating diverse populations of agents, we recommend referring to [45]. This review paper provides a detailed exploration of methods and techniques for promoting diversity among agents in multi-agent systems with focus on Ad-Hoc Teamwork.

2.2.3 Types of Multi-Agent Environments

2.2.3.1 Cooperative Setting

Cooperative Multi-Agent Reinforcement Learning is a field of research that focuses on developing algorithms and strategies for enabling a group of agents to work together towards a common goal. In cooperative MARL, the agents collaborate and coordinate their actions to maximize the overall performance or achieve a shared objective.

Unlike single-agent reinforcement learning, where a single agent interacts with an environment, cooperative MARL involves multiple agents interacting with the environment simultaneously. Each agent has its own observation of the environment, and they typically communicate and exchange information

to make joint decisions. The agents may have different capabilities, roles, or actions, but they cooperate by sharing information, coordinating their actions, and learning from each other's experiences.

The key challenges in cooperative MARL include learning effective communication and coordination strategies, dealing with the curse of dimensionality as the number of agents increases, handling partial observability, non-stationarity due to multiple learning agents, and addressing issues of coordination and cooperation in dynamic and uncertain environments.

Researchers in cooperative MARL [50, 46, 34, 57] propose various algorithms and techniques to tackle these challenges, such as centralized training with decentralized execution, decentralized learning with communication protocols, and hierarchical approaches. These algorithms aim to optimize the joint behavior of the agents by learning individual policies that align with the global objective or by learning a joint policy directly.

A fully cooperative multi-agent reinforcement learning task can be modeled as a decentralized partially observable MDP (Dec-POMDP). The Dec-POMDP can be defined by the tuple $\{S, U, P, r, Z, O, n, \gamma\}$, where:

- S is the state space of the environment
- $z^i \in Z$ is the local observation of each agent sampled according to the observation function $O(s, i) : S \times \mathcal{A} \to Z$
- The action-observation history for an agent i is denoted as $\tau^i \in T \equiv (Z \times U)^*$, and the policy $\pi^i(u^i|\tau^i) : T \times U \to [0, 1]$ of each agent is conditioned on it
- At each time step t, every agent $i \in \mathcal{A} \equiv \{1, \ldots, n\}$ chooses an action $u^i \in U$ with a decentralized policy $\pi^i(\cdot|\tau^i)$ using only its local action-observation history τ^i
- The agents jointly optimize the discounted accumulated reward

  $$J = \mathbb{E}_{s_t, \mathbf{u}_t}\left[\sum_t \gamma^t r(s, \mathbf{u})\right],$$

  where the joint action space $\mathbf{u} \in \mathbf{U} \equiv U^n$ can be denoted as a tuple $\mathbf{u} = (u^1, \ldots, u^n)$. When $n = 1$, the problem becomes a POMDP and is significantly easier to solve
- $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$ is the state transition function
- $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$ is the reward function shared by all agents
- $\gamma \in [0, 1)$ is the discount factor

The state-value function conditioned on the joint policy $\boldsymbol{\pi}$ is defined as:

$$V^{\boldsymbol{\pi}}(s_t) = \mathbb{E}_{\mathbf{u} \sim \boldsymbol{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|\, \mathbf{s}\right].$$

A collaborative team aims to learn an optimal joint policy $\boldsymbol{\pi} = \Pi_{i=1}^n \pi^i$ that maximizes the accumulated reward J.

2.2.3.2 Competitive Setting

Fully competitive multi-agent environments in reinforcement learning are typically characterized as zero-sum games. In zero-sum games, the total utility or reward available in the environment remains constant, meaning that any gain by one agent comes at the expense of another agent. This zero-sum nature adds an extra layer of complexity to the competitive setting, as the success of one agent is directly tied to the failure or reduced performance of other agents.

In fully competitive environments, the pursuit of individual rewards by each agent creates a competitive landscape where agents must strategically allocate their resources, make tactical decisions, and engage in adversarial interactions to gain an advantage over their opponents. Agents aim to maximize their own utility or minimize their opponents' utility, resulting in a dynamic and intense competitive scenario.

The zero-sum nature of fully competitive environments poses unique challenges for reinforcement learning algorithms. Agents must not only learn effective policies to achieve their objectives but also anticipate and counter the strategies employed by their opponents. This requires agents to engage in sophisticated decision-making, exploitation of weaknesses in opponents' policies, and adaptive learning to stay competitive in the environment.

2.2.3.3 Mixed Setting

Mixed setting environments in reinforcement learning refer to scenarios where the environment exhibits both cooperative and competitive aspects, requiring agents to balance their focus between cooperation and competition. In these environments, agents must navigate a complex landscape where collaboration with some agents is beneficial, while simultaneously competing against others.

In mixed settings, agents encounter situations where cooperation with certain agents can lead to mutually beneficial outcomes or shared rewards. At the same time, they must be cautious of other agents who may have conflicting objectives and pose a competitive threat. This combination of cooperative and competitive aspects introduces additional strategic considerations and decision-making challenges for the agents.

2.3 Spectral Normalization

A function is $k-$Lipschitz continuous in l_2-norm if

$$\|f(x_1) - f(x_2)\|_2 \leq k\|x_1 - x_2\|_2. \tag{2.19}$$

Considering a feed-forward layer, the Lipschitz constant of the layer is defined as the largest singular value of the weight matrix of that layer. Spectral normalization normalizes the weight matrix by its largest spectral value, constraining that layer to be $1-$Lipschitz smooth.

$$\hat{W} = \frac{W}{\|W\|} = \frac{W}{\sigma_{\max}(W)}. \tag{2.20}$$

We can also control the smoothness of the function to be k Lipschitz smooth by adding an extra parameter k which can be tuned.

$$\hat{W} = \frac{W}{\max(\sigma_{\max}(W), k)}. \tag{2.21}$$

The Lipschitz constant of a composite of two functions f_1 and f_2 with Lipschitz constant k_1 and k_2 will be bounded by $k_1 \cdot k_2$. Similarly the Lipschitz constant of a neural network can be bounded by the product of Lipschitz constant of each layer. For more details on Lipschitz constant of various layers and activation functions, we refer to [17].

2.3.1 Optimization effects of Spectral Normalization

Let us analyze the activation calculation of a feed-forward layer with and without spectral normalization. The equation for a layer i without spectral normalization can be written as:

$$z_i = W_i a_{i-1} + b_i \tag{2.22}$$

$$a_i = \mathrm{ReLU}(z_i), \tag{2.23}$$

where $a_0 \triangleq x$ is the input to the network.

Now let us look at the equations when we apply spectral normalization to a feed-forward layer.

$$\hat{z}_i = \hat{W}_i a_{i-1} + b_i \tag{2.24}$$

$$\hat{a}_i = \mathrm{ReLU}(\hat{z}_i), \tag{2.25}$$

Here $\hat{W}_i = k_i^{-1} W_i$ is the weight matrix after applying spectral normalization. Here k_i is the largest singular value of the weight matrix. Comparing the above equations, we can observe that only the weight matrix is scaled using the largest singular value, whereas the bias is unchanged. Due to this, the sign of the pre-activations z is not preserved ($[z_i > 0] \neq [\hat{z}_i > 0]$). Hence we cannot write a direct relation between $\frac{\partial \mathcal{L}}{\partial W_i}$ and $\frac{\partial \hat{\mathcal{L}}}{\partial W_i}$

For simplicity of analysis let us consider the network without bias. So the equation for a specific layer i can be written as follows:

$$z_i = W_i a_{i-1} \tag{2.26}$$

$$a_i = \mathrm{ReLU}(z_i), \tag{2.27}$$

where $a_0 \triangleq x$ is the input to the network.

Let a subset of layer $\mathcal{S} \subseteq \{1, 2, \ldots, L\}$ are spectral normalized and are individually 1-Lipschitz continuous. The weight matrix of the regularised layers can be defined as $\forall i \in \mathcal{S} : \hat{W}_i = \langle k_i^{-1} \rangle W_i$ where $k_i = \sigma_{\max}(W_i)$ is the largest singular value of that weight matrix. Here $\langle \cdot \rangle$ is the gradient stop operator and hence back-propagation is not applied through the singular value calculations.

Now let us update the equations for the above described feed-forward network when applying spectral normalization to it.

$$\hat{z}_i = k_i^{-1} W_i \hat{a}_{i-1} \tag{2.28}$$

$$\hat{a}_i = \mathrm{ReLU}(\hat{z}_i), \tag{2.29}$$

where $k_{i:j}^{-1} \triangleq \prod_{i \leq l \leq j \wedge l \in \mathcal{S}} k_l^{-1}$. We can write eq. 2.28 in terms of non-regularised activation as follows

$$\hat{z}_i = k_{1:i}^{-1} W_i a_{i-1}. \tag{2.30}$$

The above equation is valid as spectral normalization is scaling operation and hence the sign of the activation will be preserved ($[a_i > 0] = [\hat{a}_i > 0]$).

The loss is calculated on the final layer of the network and hence can be written as $\mathcal{L} \triangleq \mathrm{loss}(z_L)$. The loss calculation for the regularised network will be updated to $\hat{\mathcal{L}} \triangleq \mathrm{loss}(\hat{z}_L) = \mathrm{loss}(k_{1:L}^{-1} z_L)$.

$$\text{MLP} \qquad\qquad \text{SN-MLP} \tag{2.31}$$

$$\mathcal{L} \triangleq \mathrm{loss}(z_L) \qquad\qquad \hat{\mathcal{L}} \triangleq \mathrm{loss}(\hat{z}_L) \tag{2.32}$$

$$\frac{\partial \mathcal{L}}{\partial W_i} = J_i \delta_L a_{i-1}^T \qquad\qquad \frac{\partial \hat{\mathcal{L}}}{\partial W_i} = k^{-1} J_i \hat{\delta}_L \hat{a}_{i-1}^T, \tag{2.33}$$

where $k^{-1} = \Pi_{i \in \mathcal{S}} k_i^{-1}$, $\delta_L \triangleq \frac{\partial \mathcal{L}}{\partial z_L}$ is the Jacobian w.r.t. the network's output and similarly $\hat{\delta}_L \triangleq \frac{\partial \hat{\mathcal{L}}}{\partial \hat{z}_L}$ is the Jacobian with respect to the regularised network's output and

$$J_i \triangleq \Pi_{j=i}^{L-1} [\mathrm{diag}([z_j] > 0) W_{j+1}^T].$$

Based on the above equations, it is evident that applying spectral normalization leads to gradient scaling by k^{-1}. This shows that the optimization step of the regularised network is scheduled based on the product of largest spectral values of the normalized layers. For detailed analysis of how spectral normalization effects various layers, activation and the bias terms, we refer to [16, 17]. We also note here that spectral normalization is a form of preconditioning [6, 22, 12, 11, 27, 25, 26, 28, 42].

Chapter 3

Effects of Spectral Normalization in Multi-Agent Reinforcement Learning

In this section we focus on learning under sparse rewards in on-policy cooperative multi-agent reinforcement learning algorithms. We hypothesize that by introducing sparsity in the reward, critic learning gets affected and propose to regularise critic with spectral normalization to aid the critic to learn. Applying spectral normalization constraints the Lipschitz constant of the layers.

We empirically study the effects of reward sparsity on critic learning in MAPPO on two different cooperative multi-agent benchmarks: StarCraft multi-agent challenge (SMAC) [47] and multi-robot warehouse (RWARE) [39]. We start by comparing the performance of MAPPO with critic regularised MAPPO. We then analyze the critic learning by comparing the logarithm of the gradient norms of the critic of the two variants and show how applying spectral normalization on the critic helps stabilize its gradients. Our results help us understand the importance of critic learning in multi-agent scenarios under sparse rewards.

Our contributions can be summarised as follows:

- We introduce a sparse reward configuration for SMAC and show that it is difficult to learn when compared to standard reward configuration
- We propose to regularise the critic with spectral normalization and show that it helps learn better policies under sparse rewards
- We analyse the effects of applying spectral normalization and show that it helps 1) stabilise the critic gradients and 2) has an optimization effect of scaling the gradients of the entire critic by the product of the largest spectral value of the weight matrices

3.1 Related Works

There has been considerable development in cooperative multi-agent reinforcement learning in recent years[46, 50, 34, 57, 36]. Value-based as well as policy-based CTDE-MARL algorithms are effective

in cooperative tasks. But these works do not focus on the sparse reward scenarios and mainly address learning decentralized agents with factored value functions or policies.

Spectral Normalization has been used in GANs[37], as a regularizer which leads to better sample efficiency [17] or to improve robustness of uncertainty estimates [31]. In the context of RL, SN has been used in model-based RL in uncertainty estimation [58] to enable deeper networks [3] and to also show that SN regularised networks can compete with algorithmic innovations [16]. Tesseract [36] uses tensor decompositions to learn robust estimates for the underlying MDP dynamics and action-value function with provable sample efficiency in multi agent setting.

To the best of our knowledge, we are the first to apply SN in the context of multi-agent RL. Our work differs from the previous works in the sense that we show that SN can be used to make critic more robust to the noise induced by sparse rewards under multi-agent scenarios. Previous works have shown that adding SN to the value function estimator helps stabilize its learning by stabilizing its gradients [3] as well as act as an update-step scheduler [16]. In our work, we observe that applying SN in multi-agent scenarios leads to both of these benefits.

3.2 Experimental Setup

We use MAPPO as our on-policy multi-agent algorithm to perform all the evaluations. Implementation and configuration from [39] are used for all our experiments. The actor consists of 3 layers with GRU as the middle layer, and the critic uses 3 layered feed-forward network. All the layers have 64 neurons, and the hidden dimension of GRU is 64. Adam [23] optimizer with a learning rate of 5×10^{-4} is used for updating the network weights. Gradient clipping is applied to both the actor and critic gradients with a gradient norm 10. The weights of the actor and critic are shared across all agents[57].

For learning the critic we use 10-step temporal difference learning rule. The actor is optimized using the standard PPO objective. We normalize the returns for critic for two of our variants, FullSN-MAPPO and LastSN-MAPPO.

We test three different variants with spectral normalization on critic and a standard MAPPO:

- **FullSN-MAPPO**: Spectral Normalization (SN) is applied on all critic layers
- **MidSN-MAPPO**: SN only applied on the second layer or the middle layer of the critic
- **LastSN-MAPPO**: SN applied to the final layer of the critic
- **MAPPO**: Standard MAPPO implementation with no spectral normalization

Figure 3.1 Illustration of an RWARE environment and a SMAC map

3.3 Results

We empirically evaluate our results on two cooperative multi-agent benchmarks, multi-robot warehouse (RWARE) and starcraft multi-agent challenge (SMAC). We report our scores averaged across four seeds.

3.3.1 RWARE Environment

RWARE is a partially observable sparse reward benchmark introduced in [39]. It is a grid-world environment where the agents are rewarded for delivering the requested shelf from the warehouse. Agents can only observe a 3×3 grid surrounding themselves. We consider three different tasks which vary the grid size and the number of agents. This is a relatively simpler environment where a single agent can complete the task without any help from the other agents in the environment. This reflects in our results in Fig. 3.2 where two variants, MAPPO and MidSN-MAPPO, show similar final performance, with MidSN-MAPPO being quicker to converge.

We compare three different RWARE environments with a varying number of agents and environment sizes.

- **tiny-2ag** is the smallest map with two agents. We observe that the spectral normalized variant converges a bit faster comparatively
- **tiny-4ag** is the same as the previous map but with four agents. In this case, we do not see any significant difference between the two variants. Though our variant with normalized critic seems to converge a bit faster again
- **small-2ag** is a larger map with almost double the number of shelves in the environment with only two agents

Overall in all three environments, we observe our variant to converge early, but the final performance is almost the same.

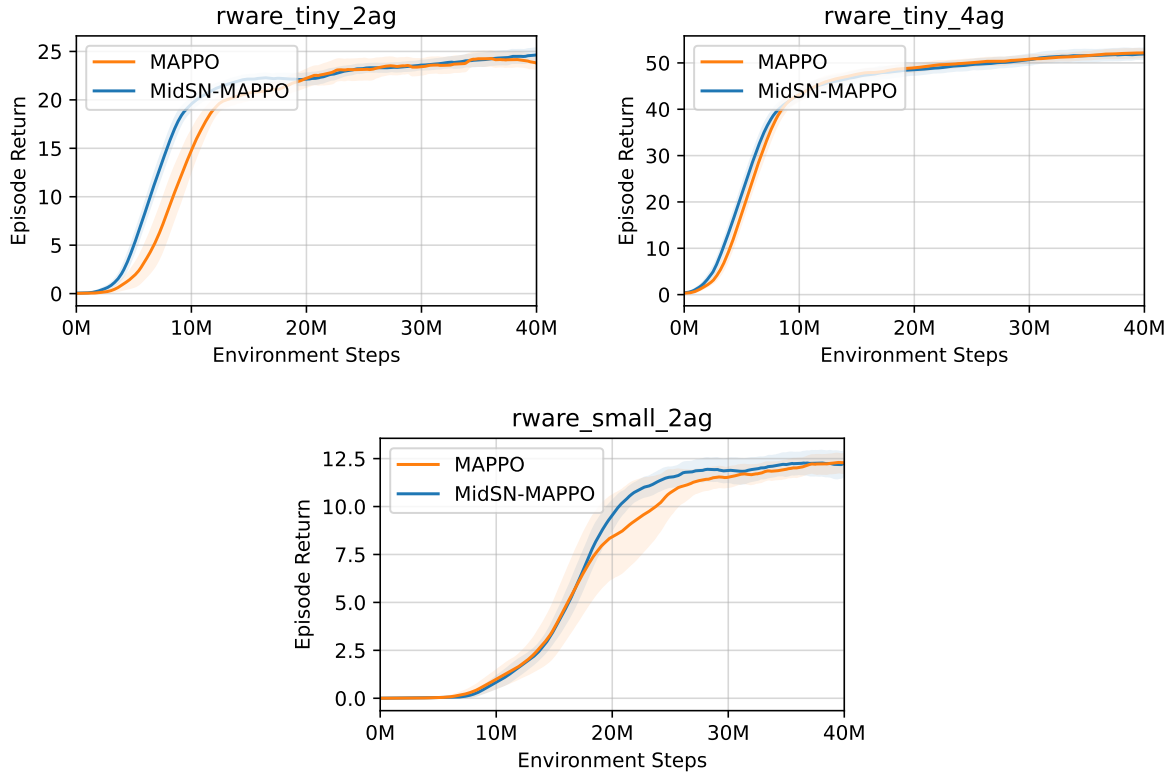

Figure 3.2 Learning curves on RWARE comparing MAPPO and MidSN-MAPPO. As RWARE is a relatively simple environment where explicit coordination is not necessary, the final performance of both variants is almost the same. However, we can observe that MidSN-MAPPO converges a bit faster.

3.3.2 SMAC Environment

SMAC is a benchmark based on the Starcraft II game. This environment consists of battle scenarios where a team of agents is controlled to defeat the enemy team, which uses fixed policies. This is also a partially observable environment where each agent only observes a fixed around itself for other agents. Here too, we consider three different tasks with a varying number of agents and unit types. The primary challenge in these tasks is learning optimal behaviour under partial observability and the large joint action space growing based on the number of agents. As we specifically wanted to evaluate the performance on sparse rewards, we propose a custom reward configuration where the agents are awarded rewards only in cases of death and win/loss. For each death in the ally team, a reward of -10 is awarded, and for each kill in the enemy team, a reward of $+10$ is awarded. Along with the death reward, a reward of $+200$ is awarded for winning the battle, killing all the enemy units, and similarly, a reward of -200 is awarded if all the units in the ally team die. We do not use rewards based on health loss due to attacks which are usually used.

We consider three **super-hard** scenarios from Starcraft Multi-Agent Challenge (SMAC) for our comparisons. Each scenario evaluates different aspects of the environment. *3s5z_vs_3s6z* helps us evaluate

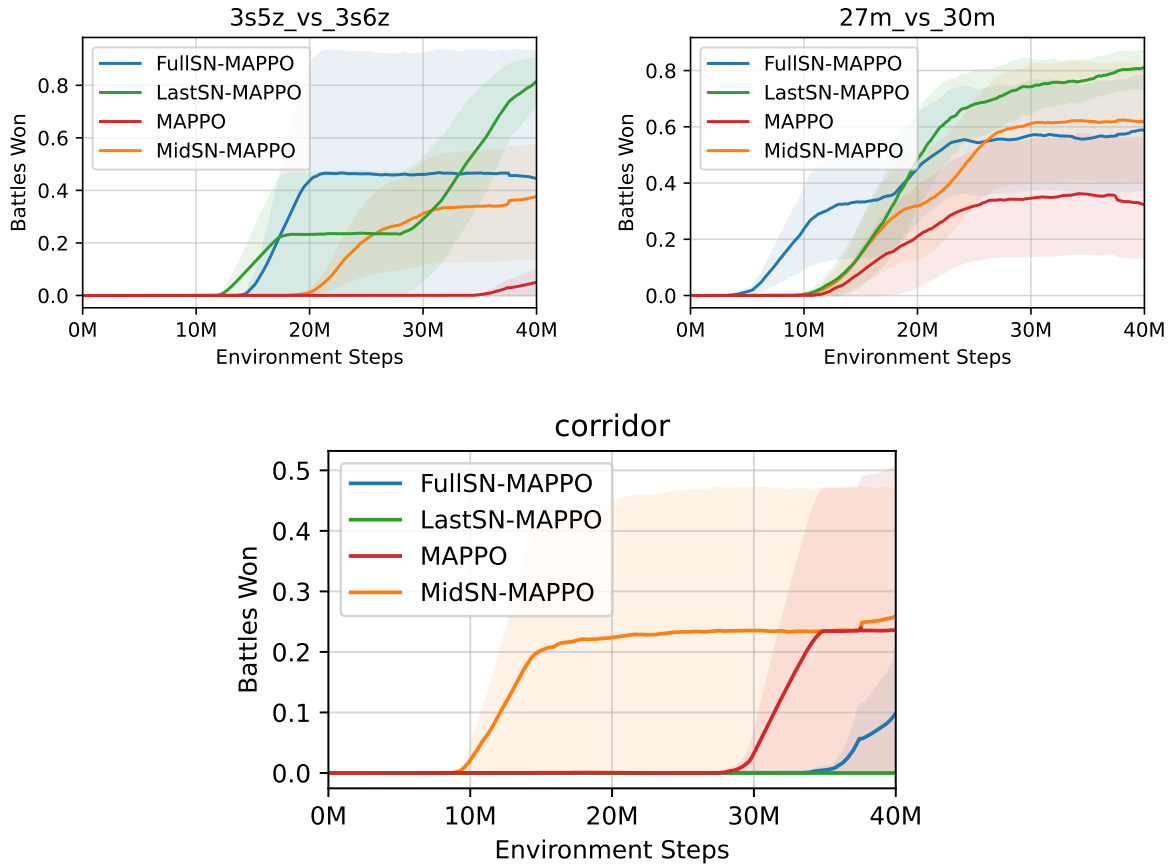

Figure 3.3 Average battles won on various SMAC maps averaged across several seeds. SMAC is a very challenging benchmark where each map requires a specific skill to be acquired to win. We observe that LastSN-MAPPO shows much better and more stable final performance than MAPPO when evaluating under sparse reward configuration. However, the results on Corridor are a bit surprising. We talk about that in more detail in Section 3.3.2

the performance of imbalanced teams. We can observe that variants with spectral normalized critic gain significant performance compared to the standard critic variant. *27m_vs_30m* has the largest ally team of 27 marines. In this scenario as well, we observe that our variant performs significantly better. This shows that our method can scale to a large number of agents. Even though spectral normalization constraints the critic, the shared weights can learn representation for many agents. *corridor* requires effective use of terrain features and block the choke point to avoid attacks from different directions. Subtle tactics like blocking the choke point to avoid attack from different directions as there is a considerable imbalance in the team since six friendly Zealots face 24 enemy Zerglings. All variants find it challenging to solve this environment consistently under sparse rewards. But still, the convergence of MidSN-MAPPO with normalized critic is quick compared to the standard variant. When we compare the number of dead enemies in Fig. 3.4, we can see that MidSN-MAPPO is performing relatively better. Even though both the algorithms fail to have high win-rates due to slow regenerative ability of enemy Zerglings, which

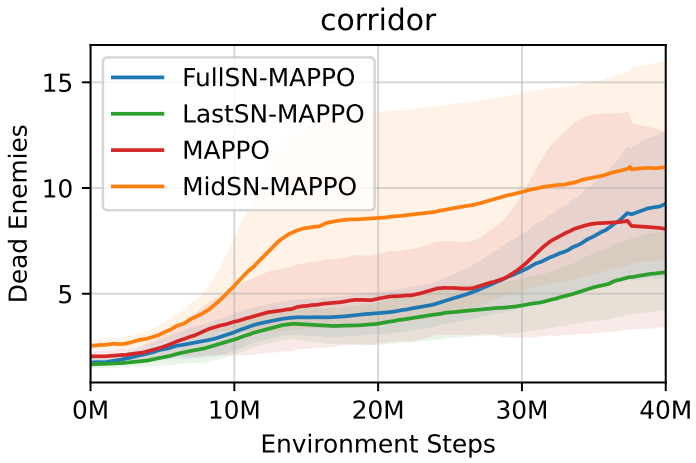

Figure 3.4 Comparing dead enemies through the training shows that MidSN-MAPPO is always ahead of MAPPO even though the final win rate is the same for the two variants, MidSN-MAPPO is able to kill more enemies even in the battles which are not a conclusive win.

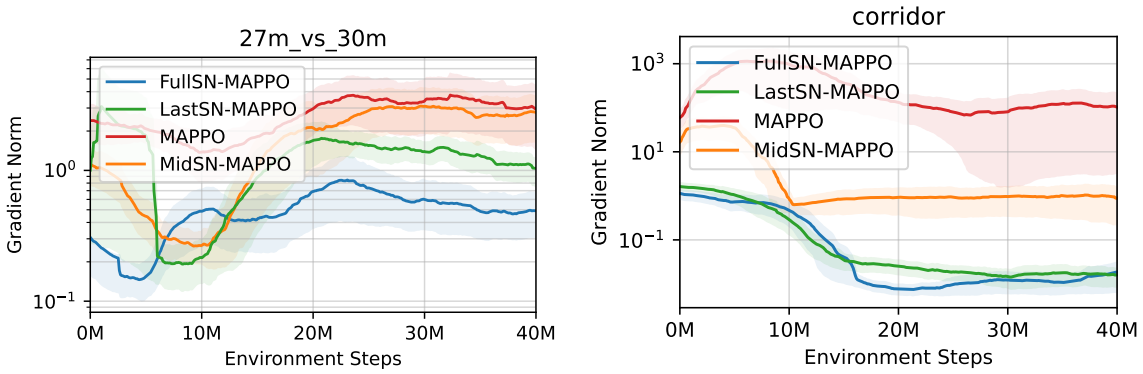

Figure 3.5 Gradient norm of the critic throughout the training. Even though MidSN-MAPPO shows improvement over MAPPO in both the environments, the reason for this performance gain is different in the two maps. In *27m_vs_30m*, the critic gradients are stable for both the variants, still MidSN-MAPPO is better. In this case, the performance gain can be attributed to the optimization effects of SN on the critic. While in *corridor*, SN helps stabilise the critic gradients, directly correlating to overall performance gain.

makes it difficult to kill them unless attacked continuously, we observe that MidSN-MAPPO is able to kill more enemies than MAPPO.

Fig. 3.3 compares the win rate on different SMAC scenarios under sparse rewards. We can observe that all three SN variants perform better than the normal MAPPO on *3s5z_vs_3s6z* and *27m_vs_30m*. LastSN-MAPPO achieves the best final win-rate consistently across various seeds. This shows that regularizing the critic with spectral normalization does indeed help to learn under sparse reward scenarios. However, the results on *corridor* paint a different picture. We observe that both the variants where SN is applied on the last layer of the critic underperform compared to the other two scenarios.

Applying SN on the last layer of critic causes its output to be smooth [16]. However, the value function doesn't need to be smooth. That is, when the focal agent has more health and the enemy agent has relatively less health, the return will be highly positive, but just a slight difference in the health of the two agents leading to an enemy agent having higher health would lead to highly negative reward. The scenarios *3s5z_vs_3s6z* and *27m_vs_30m* where FullSN-MAPPO and LastSN-MAPPO perform well have open maps and there is a lot of place for the agents to move around. Hence the value function would be smooth. However, in *corridor*, there are choke points that constraint the movements of the agents. This leads to non-smooth value function, which ultimately causes the failure of FullSN-MAPPO and LastSN-MAPPO on this scenario. It would be safe to conclude that applying SN on the final layer only helps when the value function is smooth. Otherwise, we have to restrict ourselves to not apply SN on the final layer of the critic.

3.3.2.1 Effects of Normalizing the Critic

To understand more about the effects of normalizing critic, we analyze the norm of the gradients of critic. Fig. 3.5 compares the gradient norm on two SMAC scenarios, *27m_vs_30m* and *corridor*. We observe that learning happens in both the maps, but there is a critic gradient explosion in the normal variant on *corridor*. Notice that the plots are in $\log$ scale. This shows that regularising critic with spectral norm helps stabilize the learning in critic by stabilizing its gradients.

But another question that remains is what exactly causes the performance gain in *27m_vs_30m*? As we observe, the gradient norm of both variants is almost in the same range. The performance gain, even when the gradient norm is not exploding, can be explained based on the effects of SN discussed in section 2.3.1. Let's look at the output and gradient equations of a three-layered fully-connected network. We observe that applying spectral normalization on a layer is equivalent to scaling the gradients of the complete network by the inverse of maximum spectral value ρ^{-1}. This scaling of the gradient effect acts as a step-size scheduler based on the spectral values of the regularised layers. Hence the performance gain in *27m_vs_30m* can be attributed to the gradient scaling effect of SN.

We can conclude from the above analysis that the benefits of applying spectral normalization to the critic are as follows

- Stabilise critic by constraining the gradients

- Optimization effect by scaling the gradient by the inverse of the maximum spectral value
- Better learning of smooth value functions by applying SN on the last critic layer

3.4 Limitations

It is important to note that even though SN can help in stabilizing the critic learning, it can only help up to an extent and under the conditions that the agent is able to reach some rewarding state by random exploration. In case of extremely sparse rewards, e.g., only win/loss reward in SMAC, it is extremely unlikely that the team of agents randomly stumbles upon a winning situation. As there is a very slim chance of getting an actual positive reward, there is no information presented to the critic that it can leverage. Hence stable critic helps only under the condition that the agent is able to reach rewarding states, but the reward signal might get suppressed by the noise from the untrained critic.

Chapter 4

marl-jax

Multi-agent reinforcement learning (MARL) is an important framework for training autonomous agents that operate in dynamic environments with multiple learning agents. Many potential real-world applications require the trained agents to cooperate with humans or agents not seen during training. That is, they should be able to zero-shot generalize to novel social partners. Most of the existing MARL frameworks [48, 40, 49, 59, 20] are either designed for cooperative MARL research or naively extend existing single-agent RL frameworks to work with multiple agents.

On the contrary, marl-jax is designed specifically for multi-agent research and facilitate the training and assessment of the generalization capacities of multi-agent reinforcement learning (MARL) algorithms when facing new social partners. We utilize the functionalities of JAX [4] including autograd, vectorization through *vmap*, parallel processing through *pmap*, and compilation through *jit*, resulting in highly optimized training for multiple agents.

Inspired by Acme [19], we share a lot of design philosophies with it. Reverb [8] is used as a data-store server for the replay buffer. Launchpad [56] is used for distributed computing. We use JAX [4] as the numerical computation backend for neural networks. We use *dm-env* API as our environment interaction API and extend it for multi-agent environments.

4.1 Reproducibility in Reinforcement Learning

The RL community has developed several frameworks targeting various aspects such as implementation simplicity, ease of adaptation and scaling deep RL agents. In marl-jax, we focus on ease of experimentation and adaption for training a population of agents in multi-agent environments.

A number of libraries that concentrate on single-agent reinforcement learning have been created, such as stable-baselines3 [44], dopamine [9], acme [19], RLlib [29], and CleanRL [21]. These libraries prioritize features like modularity by providing useful abstractions, ease of use by requiring minimal code to get started, distributed training and ease of comprehension and reproducibility. Other libraries such as Reverb [8], rlax [2], and launchpad [56] concentrate on specific components of an RL system.

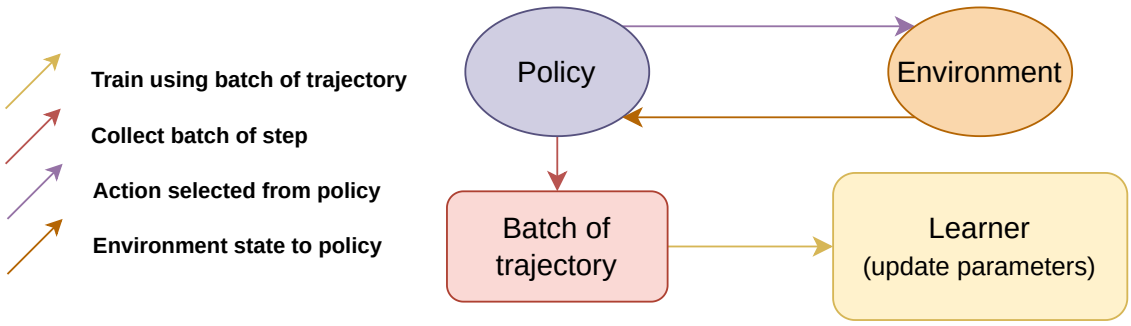

Figure 4.1 Single-threaded RL training pipeline

For multi-agent reinforcement learning, several libraries have been developed, including PyMARL [48], epymarl [40], RLlib [29], Mava [41], and PantheonRL [49]. RLlib and PantheonRL enhance existing single-agent RL algorithms to enable multi-agent training, while Mava, PyMARL, and epymarl are specifically designed for MARL but only support cooperative environments.

The advancements of Reinforcement Learning (RL) algorithms have been greatly influenced by libraries providing a range of environments. Single agent RL has been aided by libraries such as OpenAI Gym [5] and dm-env [38], which established the framework for environment interactions. Multi-agent RL has been supported by SMAC [48] and PettingZoo [54]. Recently, DeepMind has contributed to the field by open-sourcing MeltingPot [1], a library for evaluating multi-agent generalization to new social partners at scale. Similarly, efforts for measuring generalization in cooperative multi-agent settings [35] are being supported by libraries like [13].

Several recent works [19, 21, 2, 41] have begun utilizing JAX [4] due to its various benefits. These benefits include auto-vectorization, just-in-time compilation, and easy multi-GPU scaling. Therefore, we have selected JAX as the framework for our library based on its demonstrated advantages and ability to meet the computational needs of MARL.

4.2 System Architecture

We implement four different training architectures

4.2.1 Single Threaded

Figure 4.1 illustrates the sequential flow of operations in a single-threaded RL training process. At each step, the agent receives an observation from the environment, selects an action based on its policy, interacts with the environment, and receives a reward. The observation, action, reward, and next observation are collected to form a batch and then used to update agent's policy and value function using gradient descent. This process continues iteratively until the desired convergence or a specified number

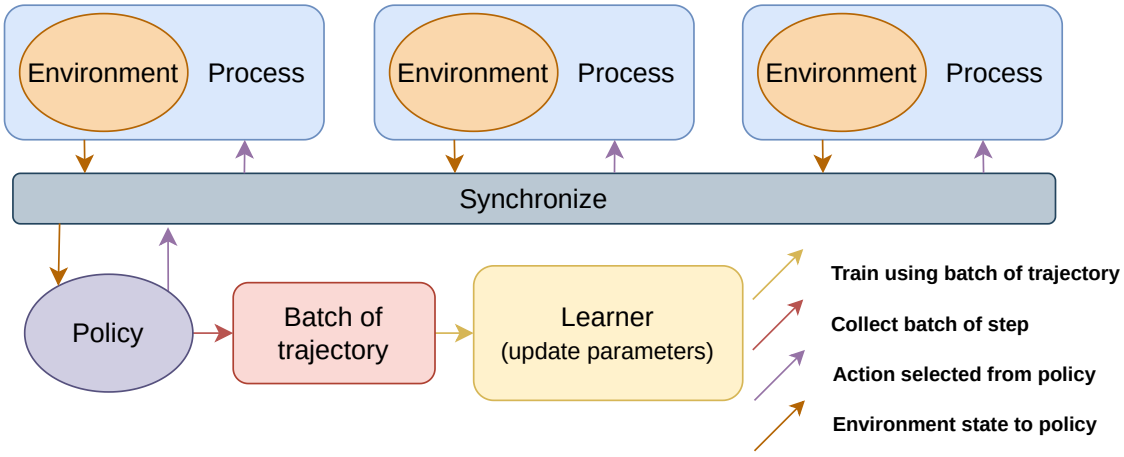

Figure 4.2 RL training with vectorized environment

of iterations is reached. Algorithm 1 describes the pseudocode for training a policy in singe-threaded manner.

```
repeat
    params = initialize params for the policy and value function
    sequence = [    ]
    timestep = environment.reset()
    repeat
        actions = predict_actions(params, timestep.observation)
        new_timestep = environment.step(actions)
        sequence.append((timestep, actions))
        timestep = new_timestep
    until not timestep.last() or len(sequence) < MAX_BATCH_SIZE
    params, logs = sgd_step(params, sequence)
    logger.write(logs)
until actor_steps < MAX_STEPS
```

Algorithm 1: Single-threaded RL training

4.2.2 Synchronous Distributed

The synchronous distributed architecture builds upon the single-threaded architecture by utilizing multiple environment instances running in parallel processes to collect a batch of experiences simultaneously. Each environment synchronously interacts with a common policy, generating sequences of states, actions, rewards, and next observations. This batch of sequences is used to update the policy and value function weights through gradient descent. By leveraging parallelization, the synchronous distributed architecture enables more efficient data collection and faster updates, leading to accelerated

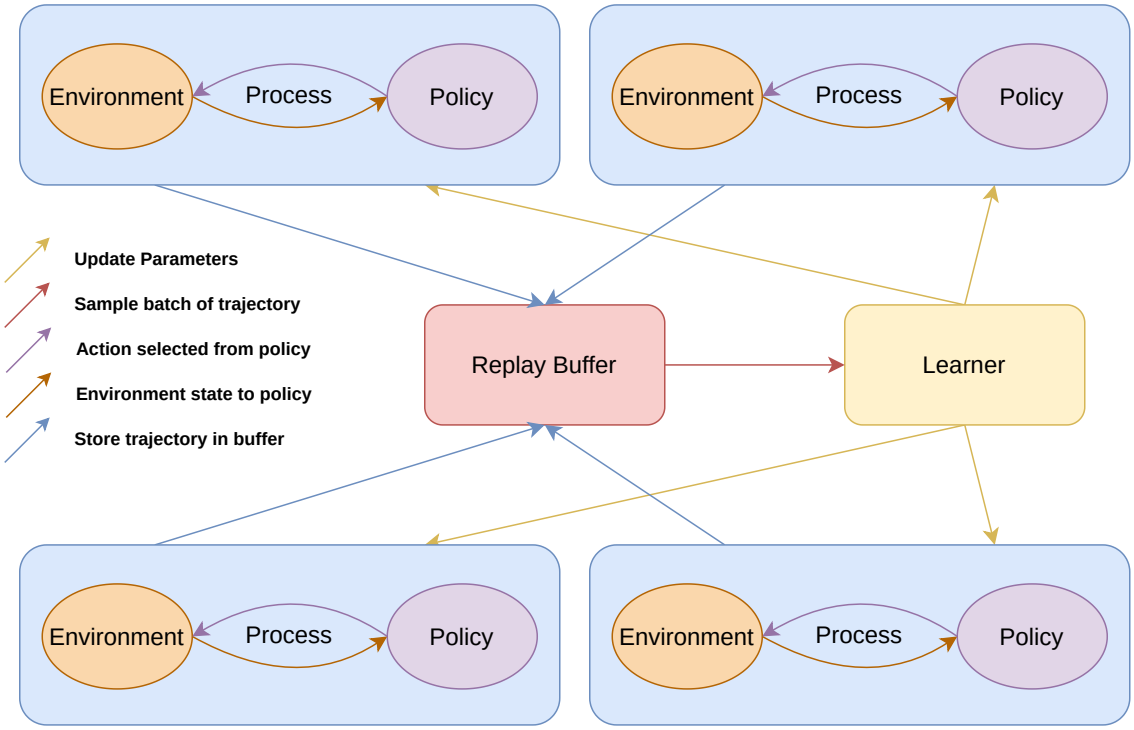

Figure 4.3 Asynchronous distributed RL training pipeline

training and improved convergence in reinforcement learning. Fig.4.2 illustrates how synchronous parallelization is achieved by running each environment instance in a separate process. The pseudocode is similar to that described in algorithm 1, with the main difference being that each interaction with the environment results in a batch of data collected from environments across all processes.

4.2.3 IMPALA-style Asynchronous Distributed

In the IMPALA-style asynchronous distributed training architecture, multiple actors run in parallel and interact with their respective environments asynchronously. Each actor collects trajectories of experience by executing its own copy of the current policy. The three main components running in parallel as separate processes are described below

- **Environment Loop:** The environment loop process interacts with the environment using the available policy and adds the collected experience to the replay buffer. Multiple parallel environment loop processes are run, each with its own copy of the environment and policy parameters. We use CPU inference for action selection on each process. To keep the policy parameters in sync with the learner process, the parameters are periodically fetched from the learner process. The action selection step is optimized using *vamp* auto-vectorization to select the action for all agents in the environment. Algorithm 2 shows the pseudocode for this process.

- **Learner:** The actual policy learning happens in this process. The learner fetches experience from the replay buffer and performs the optimization step on policy and value function parameters. We use *pmap* to auto-scale the optimization step to multiple GPUs and *vmap* based auto-vectorization to perform the optimization step for all agents in parallel. Algorithm 3 shows the pseudocode for this process.
- **Replay Buffer:** A separate process with reverb [8] server is used as a replay buffer. All the actors add experience to this server, and the learner process samples experience from the server to optimize for policy and value function parameters.

Figure 4.3 illustrates the how data flows between different components enabling asynchronous experience collection and training of the RL agent.

```
repeat
    params = fetch latest params from learner
    episode = [    ]
    timestep = environment.reset()
    repeat
        actions = predict_actions(params, timestep.observation)
        new_timestep = environment.step(actions)
        episode.append((timestep, actions))
        timestep = new_timestep
    until not timestep.last()
    replay_buffer.write(episode)
    logger.write(logs)
until actor_steps < MAX_STEPS
```

Algorithm 2: Asynchronous Environment loop

```
repeat
    batch = replay_buffer.sample()
    new_params, logs = sgd_step(params, batch)
    logger.write(logs)
until actor_steps < MAX_STEPS
```

Algorithm 3: Asynchronous Learner

4.2.4 Sebulba: Asynchronous Distributed with Inference Server

Inspired by Sebulba architecture from Podracer [18] and seed-rl [14], this architecture uses a common inference server in the asynchronous distributed architecture. When a common inference server is used in the asynchronous distributed architecture for RL training, the architecture is further enhanced to centralize the inference process. In this setup, multiple actors interact with their respective environments asynchronously, collecting trajectories of experience as before. However, instead of each actor

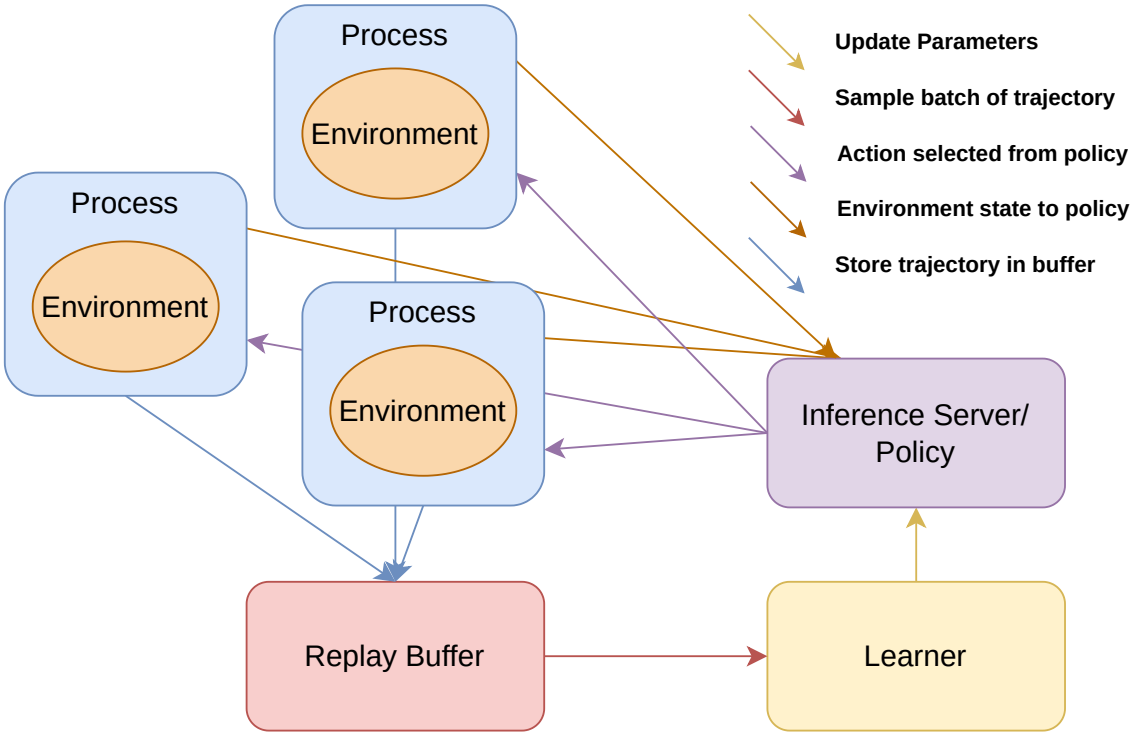

Figure 4.4 Asynchronous distributed with Inference Server

performing its own inference, they send their collected experiences to a common inference server which usually has access to hardware-accelerator such as GPU.

By utilizing a common inference server, several advantages can be achieved. First, it reduces the computational load on the individual actors, as they no longer need to perform their own inference. This enables the actors to focus on data collection, resulting in more efficient and faster interaction with the environment. Second, the use of a shared policy network ensures that all actors are making decisions based on the same set of parameters. This improves the consistency and stability of the training process, as it prevents any discrepancies that may arise from differences in local copies of the policy network.

Figure 4.4 shows the block-diagram and data-flow in asynchronous architecture with inference server. The pseudocode for learner will be exactly the same as the Algorithm 3 and for actor it is same as Algorithm 2 where the *predict_actions* functions will be a call to the inference server instead of local policy network.

4.3 Supported Environments

We support two multi-agent environment suits, which consist of simultaneous acting homogeneous agents.

4.3.1 Overcooked

The Overcooked environment [7] is a popular benchmark in the field of Multi-Agent Reinforcement Learning (MARL) that simulates a cooperative cooking scenario based on the popular game *Overcooked*. It provides a challenging and interactive environment where multiple agents collaborate to prepare dishes in a virtual kitchen.

In Overcooked, the goal is to efficiently work together as a team to prepare and serve a variety of meals. The agents control different characters within the kitchen and must coordinate their actions to complete tasks such as chopping ingredients, cooking, and delivering finished dishes to customers. Collaboration and coordination are essential to maximize efficiency and achieve high scores.

The environment features various elements that add complexity to the task. For example, the kitchen layout may include obstacles that require agents to navigate around, limited resources like cutting boards and stoves that need to be shared, and time-sensitive customer orders that must be fulfilled promptly. Additionally, agents need to strategize and communicate effectively to optimize their actions and avoid potential bottlenecks or collisions.

Overcooked is designed to test the ability of MARL algorithms to solve cooperative tasks in dynamic and complex environments. It challenges agents to exhibit skills such as coordination, planning, communication, and adaptive decision-making.

4.3.2 Melting Pot

Melting Pot [1] suite designed with the objective of evaluating generalization to novel situations and coplayers. The Melting Pot 2.0 suite consists of 50 different environments and over 256 unique test scenarios to evaluate the trained population of agents on broad range of topics such as social dilemmas, task partioning, resource sharing, etc

Melting Pot evaluation methodology is captured by the following equation:

Substrate + Background Population = Scenario

- **Substrate:** The term ”substrate” refers to the static or physical aspects of the environment in a simulation. It encompasses elements such as the layout of the map, the placement of objects, the rules governing their movement, and the physics involved. In essence, the substrate defines the stationary or unchanging components of the environment's dynamics. It sets the foundation and structure upon which other dynamic elements and interactions can take place. By defining the substrate, the simulation establishes the framework for how the environment behaves and provides a stable backdrop against which other entities and events can unfold.

- **Background Population:** The term ”background population” refers to a group of simulated entities within a simulation that have their own agency or ability to take actions and make decisions. In other words, these entities are not passive or static; they actively participate in the simulation and contribute to its dynamics. They can interact with other entities, respond to stimuli or events,

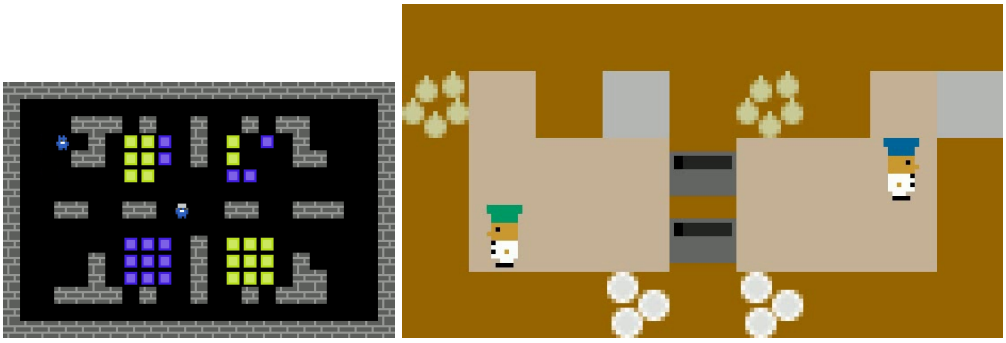

Figure 4.5 marl-jax supports two major environment suits, Meltingpot [1] and Overcooked [7]

and potentially influence the overall behavior and outcomes of the simulation. The background population adds an element of realism and complexity to the simulation, making it more dynamic and reflective of real-world scenarios.

- **Scenario:** In the context of simulation or modeling, a scenario is created by combining the substrate and the background population. The substrate refers to the static or physical part of the environment, such as the layout, objects, and physics rules. On the other hand, the background population consists of simulated entities with agency, meaning they can take actions and make decisions within the simulation.

  By integrating the substrate and the background population, a scenario is formed that represents a specific setting or situation within the simulation. The substrate provides the foundation, defining the physical attributes and constraints of the environment. This includes factors like the terrain, structures, objects, and their spatial arrangement. The substrate sets the stage for interactions and events to occur.

  The background population adds a dynamic aspect to the scenario. These simulated entities have their own behaviors, goals, and decision-making processes. They can interact with each other, respond to stimuli or events in the environment, and potentially influence the overall dynamics of the scenario. The actions and interactions of the background population create a realistic and evolving simulation environment.

  Together, the substrate and background population create a scenario that encapsulates a particular context or situation within the simulation. This scenario can be designed to simulate real-world scenarios, test hypotheses, study the behavior of complex systems, or provide a platform for experimentation and analysis. By carefully defining the substrate and background population, researchers and practitioners can create meaningful and informative scenarios that capture the intricacies of the system being studied.

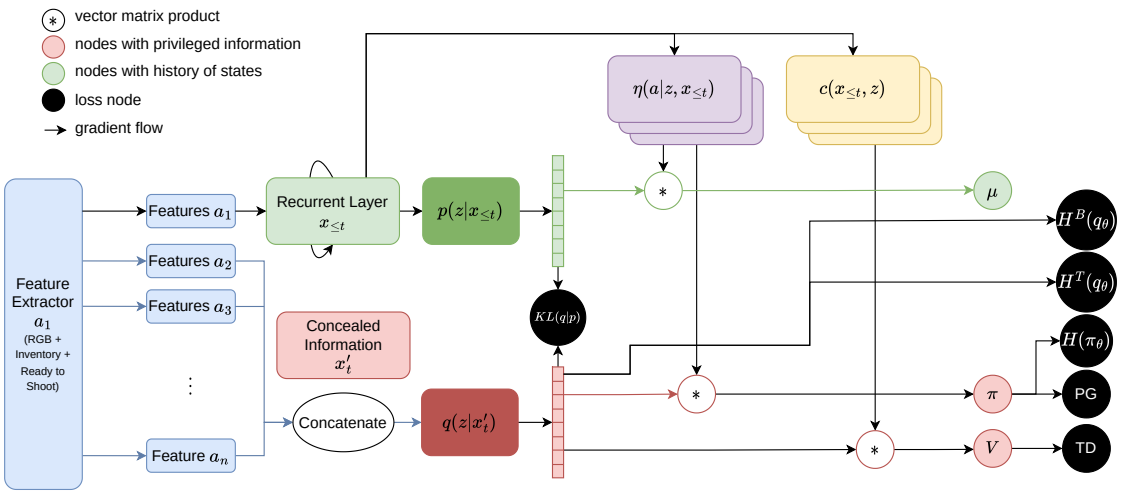

Figure 4.6 Options as Responses Architecture

4.4 Algorithms Implemented

We currently support two major algorithms

- **IMPALA:** A standard actor-critic based independent learning algorithm using V-trace [15] for off-policy corrections

- **OPRE:** Options as Responses [55] follows actor-critic based learning but its objective is specifically designed to generalize to novel partners. It is used as one of the baseline in MeltingPot [1]. We are the first to provide an open-source implementation of OPRE. The architecture illustration is shown in figure 4.6

4.5 Utilities

We provide three major utilities 1) *train.py*, 2) *evaluate.py* and 3) *evaluation_results.py*

- **train.py:** The entry point for training a population of agents in the given environment

- **evaluate.py:** Used to evaluate the generalization performance on with various partner agents

- **evaluation_results.py:** Aggregates the evaluation results by averaging across multiple seeds and presents a table

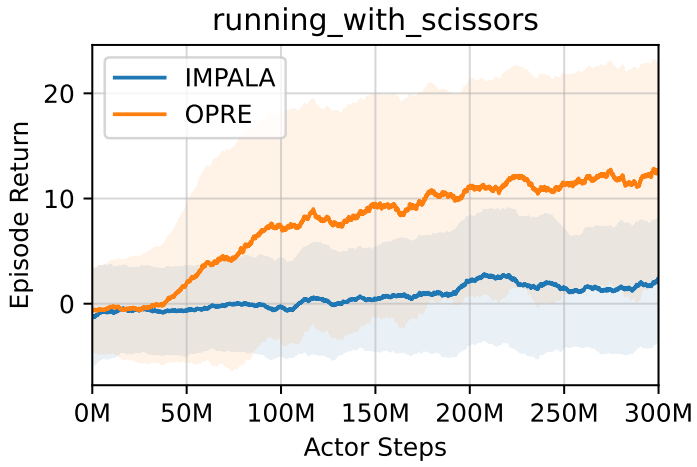

Figure 4.7 Training Plot on Running with Scissors in the matrix

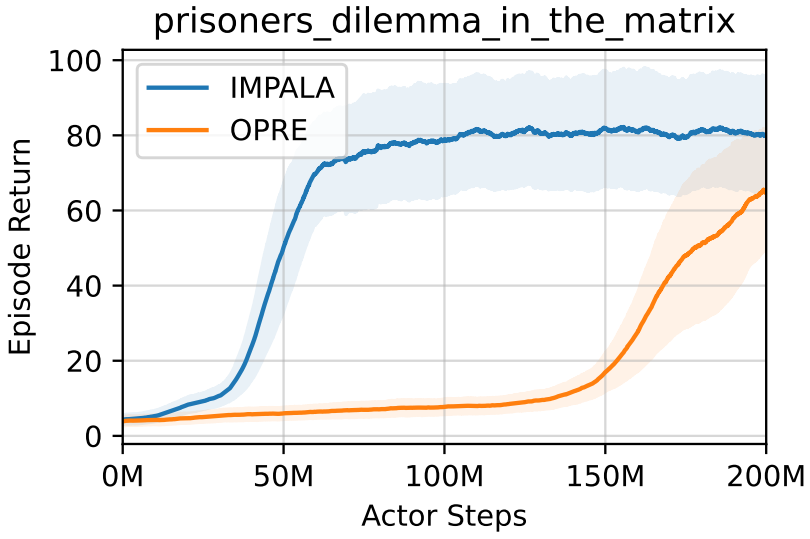

Figure 4.8 Training Plot on Prisoners Dilemma in the Matrix

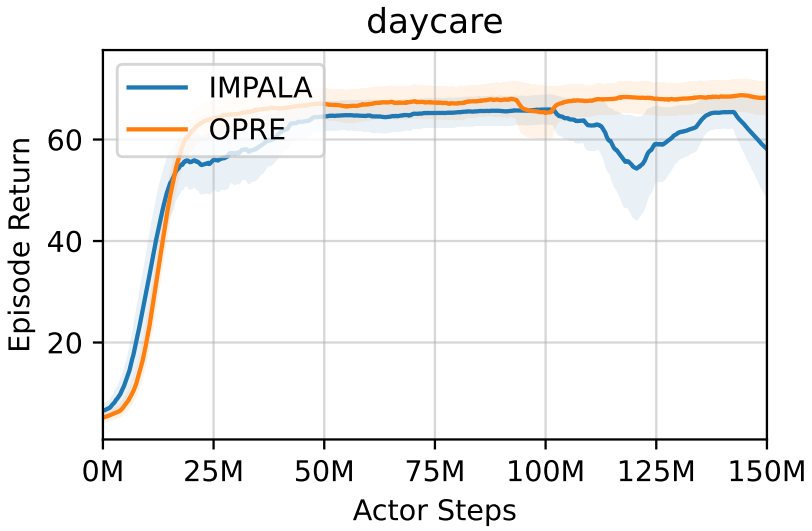

Figure 4.9 Training Plot on Daycare

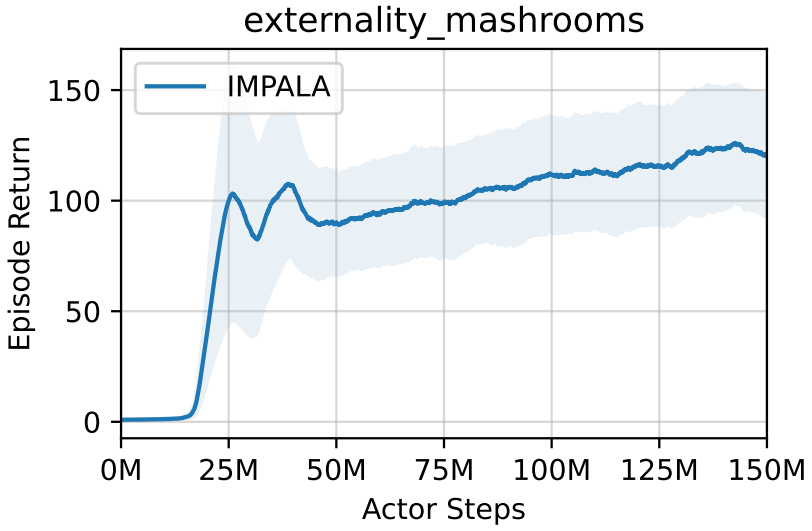

Figure 4.10 Training Plot on Externality Mushrooms

	OPRE	IMPALA
Substrate	0.00	0.00
Scenario 0	4.91	-7.10
Scenario 1	3.79	-2.65
Scenario 2	10.52	5.97
Scenario 3	13.52	11.52
Scenario 4	6.60	0.66

Table 4.1 Evaluation on Running with Scissors in the matrix

	IMPALA	OPRE
Substrate	106.85	38.18
Scenario 0	131.00	59.71
Scenario 1	176.54	114.69
Scenario 2	79.58	27.97
Scenario 3	62.80	41.76
Scenario 4	48.63	38.75
Scenario 5	65.82	47.66
Scenario 6	101.83	40.34
Scenario 7	83.33	49.82
Scenario 8	77.75	32.59
Scenario 9	78.41	74.62

Table 4.2 Evaluation on Prisoners Dilemma in the Matrix

4.6 Results

We evaluate our implementation in two environments to assess its performance and generalization capabilities across different types of multi-agent scenarios. The first environment, Meltingpot, encompasses a wide range of game types, including cooperative, competitive, and general-sum games. In Meltingpot, agents interact with each other to achieve various objectives, which can involve cooperation, competition, or a combination of both. This environment allows us to examine how well our implementation handles different types of interactions and strategies, evaluating its performance in cooperative, competitive, and general-sum settings.

The second environment, Overcooked, focuses specifically on cooperative multi-agent scenarios. In Overcooked, agents work together in a shared kitchen to prepare meals and serve customers. The emphasis in this environment is on effective coordination, communication, and cooperation among the agents to maximize efficiency and customer satisfaction. By evaluating our implementation in Overcooked, we can specifically assess its performance and effectiveness in cooperative multi-agent settings, where collaboration and teamwork are crucial for success.

	IMPALA	OPRE
Substrate	65.94	67.83
Scenario 0	0.89	0.33
Scenario 1	109.11	126.00
Scenario 2	0.22	0.00
Scenario 3	154.56	171.33

Table 4.3 Evaluation on Daycare

	IMPALA
Scenario 0	547.70
Scenario 1	13.21
Scenario 2	293.02
Scenario 3	38.04
Substrate	91.56

Table 4.4 Evaluation on Externality Mushrooms

By evaluating our implementation in both Meltingpot and Overcooked, we gain a comprehensive understanding of its performance in a range of multi-agent scenarios. This evaluation enables us to analyze how well our approach adapts to different types of interactions, strategies, and objectives, and provides valuable insights into its strengths and limitations. The findings from these evaluations contribute to advancing our understanding of multi-agent reinforcement learning and inform further research and development in this field.

4.6.1 MeltingPot

We conduct evaluations on four distinct environments from the Meltingpot-v1 and Meltingpot-v2 domains.

In Meltingpot-v1, we evaluate our approach on two environments:

- **Running with Scissors in the Matrix**: The training progress is visualized in Figure 4.7, and the evaluation scores across different episodes are provided in Table 4.1
- **Prisoners' Dilemma in the Matrix**: The training progress is depicted in Figure 4.8, and the evaluation scores for various scenarios are presented in Table 4.2

In Meltingpot-v2, we assess our approach on two additional environments:

- **Daycare**: The training progress is shown in Figure 4.9, and the evaluation scores for different scenarios are summarized in Table 4.3

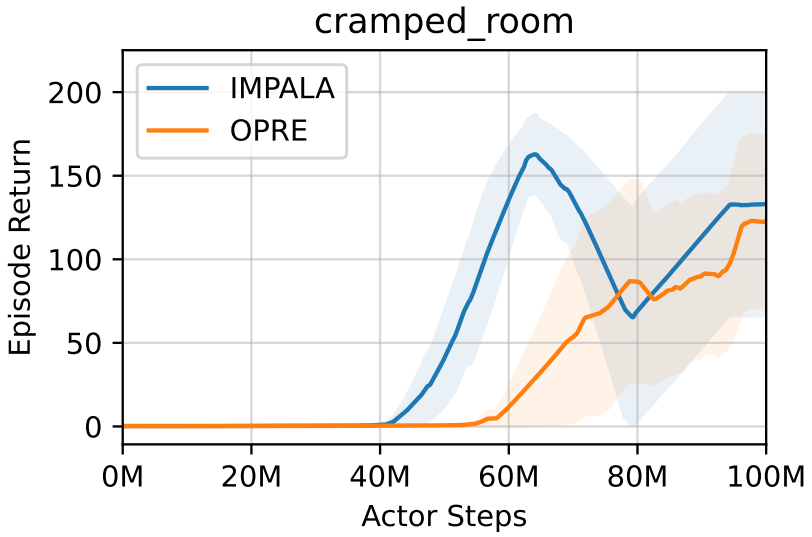

Figure 4.11 Training Plot on Cramped Room

- **Externality Mushrooms Dense**: The training progress is illustrated in Figure 4.10, and the evaluation scores for various scenarios are provided in Table 4.4

These evaluations allow us to analyze the performance of our approach across different environments and scenarios within the Meltingpot framework. The training plots provide insights into the learning progress, while the evaluation scores offer quantitative measures of the agent's performance in various scenarios.

4.6.2 Overcooked

We evaluate the performance of the algorithms on the *Cramped Room* environment from the Overcooked domain. The training progress of the algorithms is visualized in Figure 4.11, providing insights into their learning dynamics and convergence behavior.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In the first part of this thesis, we have addressed the challenging problem of learning a reliable critic in multi-agent sparse reward scenarios. The exponential growth of the joint action space with the number of agents, combined with reward sparsity and environmental noise, presents significant obstacles to achieving accurate learning. To overcome these challenges, we have proposed a novel approach of regularizing the critic with spectral normalization (SN). Our experimental results demonstrate that the regularized critic exhibits enhanced robustness, enabling faster learning even in complex multi-agent scenarios. These findings underscore the crucial role of critic regularization in achieving stable learning outcomes and provide valuable insights for future research in the field of multi-agent reinforcement learning.

In the second part of this thesis, we introduced marl-jax, a comprehensive software package specifically designed for Multi-Agent Reinforcement Learning (MARL). By leveraging DeepMind's JAX ecosystem and their RL framework, marl-jax provides a powerful and flexible platform for training and evaluating the social generalization capabilities of agents. It supports cooperative and competitive environments with multiple agents acting simultaneously, enabling researchers to study the dynamics of social interactions in complex multi-agent scenarios.

With marl-jax, researchers can easily train agent populations and evaluate their generalization performance using an intuitive command-line interface. The package offers a range of functionalities, including data collection, training, and evaluation, allowing for efficient experimentation and comparison of different MARL algorithms. By providing a reliable and standardized baseline, marl-jax serves as a valuable tool for researchers interested in investigating social generalization in MARL.

The introduction of marl-jax fills a crucial gap in the field of MARL, offering a unified and accessible framework for studying social behaviors and generalization capabilities of agents. By providing a solid foundation for further research, marl-jax contributes to the advancement of MARL algorithms and methodologies, fostering innovation and progress in the exciting field of multi-agent learning.

5.2 Future Work

In the first part of our research, we have focused on the on-policy algorithm MAPPO for sparse reward learning. However, there are opportunities to extend our work to off-policy algorithms, which could provide further insights into the challenges and potential solutions in handling sparse reward scenarios.

While our work has successfully addressed moderately sparse reward scenarios, it would be interesting to investigate the effects of highly sparse rewards where the reward signal is only obtained at the end of an episode. Exploring the applicability of our approach in such settings could provide valuable insights into handling extreme sparsity and designing more efficient learning algorithms.

Furthermore, our research has primarily focused on cooperative multi-agent reinforcement learning (MARL). It would be beneficial to extend our investigations to competitive games and general-sum games to understand how our methods and techniques translate to different types of MARL environments.

While we have evaluated the effectiveness of spectral normalization as a regularizer in our work, there is still room for exploring other regularization techniques and their impacts on MARL. Investigating alternative regularization methods could provide further improvements in stability and performance, leading to more robust learning in multi-agent scenarios.

Additionally, we would like to delve deeper into the analytical relationship between sample efficiency in reinforcement learning and the degree of spectral normalization applied. Understanding the theoretical foundations of spectral normalization and its impact on the sample complexity of MARL algorithms could provide valuable insights into the trade-offs involved in regularization.

In the second part of our research, we have developed a well-tested codebase that focuses on the reproducibility and scalability of training architectures in MARL. However, our codebase can serve as a strong foundation for further research on generalization to novel co-players in multi-agent reinforcement learning. Researchers can leverage our codebase to explore and evaluate novel generalization techniques, facilitating advancements in the field of MARL and its application in complex environments.

By addressing these future research directions, we can further enhance our understanding of sparse reward learning, regularization techniques in MARL, and the generalization capabilities of multi-agent systems.

Chapter 6

Publications

6.1 Relevant Publications

1. **Kinal Mehta** et al., *"Effects of Spectral Normalization in Multi-agent Reinforcement Learning"*, **International Joint Conference on Neural Networks (IJCNN)**, 2023

2. **Kinal Mehta** et al., *"marl-jax: Multi-agent Reinforcement Leaning Framework for Social Generalization"*, **European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) - Demo Track**, 2023

Bibliography

[1] J. P. Agapiou, A. S. Vezhnevets, E. A. Duéñez-Guzmán, J. Matyas, Y. Mao, P. Sunehag, R. Köster, U. Madhushani, K. Kopparapu, R. Comanescu, D. J. Strouse, M. B. Johanson, S. Singh, J. Haas, I. Mordatch, D. Mobbs, and J. Z. Leibo. Melting Pot 2.0, 2022.

[2] I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, J. Quan, G. Papamakarios, R. Ring, F. Ruiz, A. Sanchez, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, L. Wang, W. Stokowiec, and F. Viola. The DeepMind JAX Ecosystem, 2020.

[3] J. Bjorck, C. P. Gomes, and K. Q. Weinberger. Towards Deeper Deep Reinforcement Learning with Spectral Normalization, 2022.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[6] D. E. Carlson, E. Collins, Y.-P. Hsieh, L. Carin, and V. Cevher. Preconditioned spectral descent for deep learning. In *NeurIPS*, volume 28, 2015.

[7] M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan. On the Utility of Learning about Humans for Human-AI Coordination. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[8] A. Cassirer, G. Barth-Maron, E. Brevdo, S. Ramos, T. Boyd, T. Sottiaux, and M. Kroiss. Reverb: A Framework For Experience Replay, 2021.

[9] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018.

[10] R. Charakorn, P. Manoonpong, and N. Dilokthanakul. Generating Diverse Cooperative Agents by Learning Incompatible Policies. In *ICML 2022 Workshop AI for Agent-Based Modelling*, 2022.

[11] S. Das, S. Katyan, and P. Kumar. Domain decomposition based preconditioned solver for bundle adjustment. In *Computer Vision, Pattern Recognition, Image Processing, and Graphics*, 2020.

[12] S. Das, S. Katyan, and P. Kumar. A deflation based fast and robust preconditioner for bundle adjustment. In *WACV*, pages 1782–1789, January 2021.

[13] B. Ellis, S. Moalla, M. Samvelyan, M. Sun, A. Mahajan, J. N. Foerster, and S. Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2212.07489*, 2022.

[14] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. 2019.

[15] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018.

[16] F. Gogianu, T. Berariu, M. Rosca, C. Clopath, L. Busoniu, and R. Pascanu. Spectral Normalisation for Deep Reinforcement Learning: An Optimisation Perspective, 2021.

[17] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. Regularisation of Neural Networks by Enforcing Lipschitz Continuity, 2020.

[18] M. Hessel, M. Kroiss, A. Clark, I. Kemaev, J. Quan, T. Keck, F. Viola, and H. van Hasselt. Podracer architectures for scalable reinforcement learning. 2021.

[19] M. W. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, N. Momchev, D. Sinopalnikov, P. Stańczyk, S. Ramos, A. Raichuk, D. Vincent, L. Hussenot, R. Dadashi, G. Dulac-Arnold, M. Orsini, A. Jacq, J. Ferret, N. Vieillard, S. K. S. Ghasemipour, S. Girgin, O. Pietquin, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Friesen, R. Haroun, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, S. Srinivasan, A. Cowie, Z. Wang, B. Piot, and N. de Freitas. Acme: A Research Framework for Distributed Reinforcement Learning, 2022.

[20] S. Hu, Y. Zhong, M. Gao, W. Wang, H. Dong, Z. Li, X. Liang, X. Chang, and Y. Yang. Marllib: A scalable multi-agent reinforcement learning library. *arXiv preprint arXiv:2210.13708*, 2022.

[21] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. M. Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, 23(274), 2022.

[22] S. Katyan, S. Das, and P. Kumar. Two-grid preconditioned solver for bundle adjustment. In *WACV*, pages 3588–3595, 2020.

[23] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2017.

[24] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12:1008–1014, 2000.

[25] P. Kumar. Aggregation based on graph matching and inexact coarse grid solve for algebraic two grid. *International Journal of Computer Mathematics*, 91(5):1061–1081, 2014.

[26] P. Kumar, L. Grigori, F. Nataf, and Q. Niu. On relaxed nested factorization and combination preconditioning. *International Journal of Computer Mathematics*, 93(1):179–199, 2016.

[27] P. Kumar, K. Meerbergen, and D. Roose. Multi-threaded nested filtering factorization preconditioner. In *Applied Parallel and Scientific Computing*, volume 7782, pages 220–234. Springer, Springer, Berlin, Heidelberg, 2013.

[28] C. Li, C. Chen, D. Carlson, and L. Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks, 2015.

[29] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[31] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan. Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness, 2020.

[32] R. Lowe, YI. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[33] A. Lupu, B. Cui, H. Hu, and J. Foerster. Trajectory Diversity for Zero-Shot Coordination. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021.

[34] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. Maven: Multi-agent variational exploration. In *NeurIPS*, pages 7611–7622, 2019.

[35] A. Mahajan, M. Samvelyan, T. Gupta, B. Ellis, M. Sun, T. Rocktäschel, and S. Whiteson. Generalization in cooperative multi-agent systems. *arXiv preprint arXiv:2202.00104*, 2022.

[36] A. Mahajan, M. Samvelyan, L. Mao, V. Makoviychuk, A. Garg, J. Kossaifi, S. Whiteson, Y. Zhu, and A. Anandkumar. Tesseract: Tensorised actors for multi-agent reinforcement learning. In *ICML*, volume 139, pages 7301–7312. PMLR, 2021.

[37] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[38] A. Muldal, Y. Doron, J. Aslanides, T. Harley, T. Ward, and S. Liu. dm_env: A python interface for reinforcement learning environments, 2019.

[39] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks, 2021.

[40] G. Papoudakis, F. Christianos, L. Schfer, and S. V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

[41] A. Pretorius, K.-a. Tessera, A. P. Smit, C. Formanek, S. J. Grimbly, K. Eloff, S. Danisa, L. Francis, J. Shock, H. Kamper, W. Brink, H. Engelbrecht, A. Laterre, and K. Beguir. Mava: A research framework for distributed multi-agent reinforcement learning, 2021.

[42] Y. Qiao, B. P. F. Lelieveldt, and M. Staring. An efficient preconditioner for stochastic gradient descent optimization of image registration. *IEEE Transactions on Medical Imaging*, 38(10):2314–2325, 2019.

[43] W. Qiu, X. Ma, B. An, S. Obraztsova, S. Yan, and Z. Xu. RPM: Generalizable Behaviors for Multi-Agent Reinforcement Learning. In *International Conference on Learning Representations*, 2023.

[44] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268), 2021.

[45] A. Rahman, E. Fosong, I. Carlucho, and S. V. Albrecht. Generating Diverse Teammates to Train Robust Agents For Ad Hoc Teamwork, 2023.

[46] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning, 2018.

[47] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson. The StarCraft multi-agent challenge. volume 4. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[48] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[49] B. Sarkar, A. Talati, A. Shih, and D. Sadigh. PantheonRL: A MARL Library for Dynamic Training Interactions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11), 2022.

[50] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv:1706.05296*, 2017.

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

[52] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[53] M. Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, ICML'93, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[54] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

[55] A. S. Vezhnevets, Y. Wu, R. Leblond, and J. Z. Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent RL, 2020.

[56] F. Yang, G. Barth-Maron, P. Stańczyk, M. Hoffman, S. Liu, M. Kroiss, A. Pope, and A. Rrustemi. Launchpad: A Programming Model for Distributed Machine Learning Research, 2021.

[57] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. 2022.

[58] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. MOPO: Model-based Offline Policy Optimization, 2020.

[59] M. Zhou, Z. Wan, H. Wang, M. Wen, R. Wu, Y. Wen, Y. Yang, Y. Yu, J. Wang, and W. Zhang. Malib: A parallel framework for population-based multi-agent reinforcement learning. *Journal of Machine Learning Research*, 24(150):1–12, 2023.