Stackelberg Multi-Agent Reinforcement Learning for Hierarchical Environments

by

Sahil Pereira

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis explores the application of multi-agent reinforcement learning in domains containing asymmetries between agents, caused by differences in information and position, resulting in a hierarchy of leaders and followers. Leaders are agents that have access to follower agent policies and the ability to commit to an action before the followers. These followers can observe actions taken by leaders and respond to maximize their own payoffs. Since leaders know the follower policies, they can manipulate the followers to elicit a better payoff for themselves.

In this work, we focus on the problem of training agents in a multi-agent setting with continuous actions at different levels of hierarchy to obtain the best payoffs at their given positions. To address this problem we propose a new algorithm, Stackelberg Multi-Agent Reinforcement Learning (SMARL) that incorporates the Stackelberg equilibrium concept into the multi-agent deep deterministic policy gradient (MADDPG) algorithm. This enables us to efficiently train agents at all levels in the hierarchy. Since maximization over a continuous action space is intractable, we propose a method to solve our Stackelberg formulation for continuous actions using conditional actions and gradient descent.

We evaluate our algorithm on multiple mixed cooperative and competitive multi-agent domains, consisting of our custom built highway driving environment and a subset of the multi-agent particle environments. We show that agents trained using our proposed algorithm outperform those trained with existing methods in most hierarchical domains, and are comparable in the rest.

Acknowledgements

I would like to convey my sincere thanks to my supervisor Dr. Mark Crowley without whom my research would not have been possible. I was given the freedom and encouragement to pursue various ideas and received his support and guidance throughout the past two years.

I am also grateful to all the members of our Machine Learning Lab for fostering a welcoming and helpful environment. Their presence and support made these years of work more enjoyable.

Though my thesis took longer than expected, my parents did not waver in their love and support. I am grateful for their faith and confidence in me, and for pushing me to achieve greater goals.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is a large class of real world problems which involve interactions between multiple agents. This has resulted in growing interest in the domain of multi agent reinforcement learning (MARL), and development of algorithms that aim to train agents to achieve their goals in a shared environment. While there have been many advances in this domain inspired from game theory, there is not much work making use of game theory to address the hierarchies exhibited within these environments. In this thesis we study the hierarchy among agents, imposed by the situation within an environment, and apply game theory concepts to leverage these hierarchies for improved training.

The following sections describe hierarchical environments, present the problem description and summarize our thesis contributions.

1.1 Hierarchical Environments

Advances in reinforcement learning (RL) have been successfully applied to solve challenging problems in single agent domains such as game playing [25, 30] and robotics [19]. However, many real world problems involve interactions between multiple agents in domains such as: discovery of communication channels and language [32, 27], multiplayer games [28], analysis of social dilemmas [18] and social influence as intrinsic motivation [12]. Single agent hierarchical reinforcement learning can also be treated as a multi-agent system where multiple levels of the hierarchy are equivalent to multiple agents [6, 24].

In many multi-agent domains, hierarchies can arise due to asymmetric information between agents, allowing agents with more information to enforce their strategies onto

others. Asymmetry can also arise when all agents have equal information, but some have the ability to commit to their action before the others. This ability to commit to an action first forces other agents to react to the situation enforced onto them by reducing their space of desirable actions. In such domains, agents at the top of the hierarchy are called leaders and can influence the actions of agents at the bottom, called followers.

In non-cooperative multi-agent domains without hierarchies, it is possible to obtain a Nash equilibrium given that agents know the equilibrium strategy of other players. While in hierarchical domains, the Stackelberg equilibrium concept can be applied to obtain optimal leader-follower strategies. The Stackelberg equilibrium has been extended to reinforcement learning to solve asymmetric problems in multi-agent domains such as cognitive networks [11, 5], sequential social dilemmas [18], security games [1] and pricing problems [15, 16]. Most of the successes of applying Stackelberg equilibrium to multi-agent domains have utilized discrete action spaces, singular focused objectives or strategy mixing instead of learning. We want to extend the Stackelberg game theoretic concept to cooperative and competitive multi-agent domains with continuous action spaces.

1.2 Problem Description

Current Stackelberg approaches for multi-agent domains are not suited for environments with continuous action spaces and co-evolving strategies. A critical challenge of adapting the Stackelberg formulation is the need to maximize over the action space, which is intractable for continuous spaces since there can be an infinite number of possible actions. Another issue is that the environment becomes non-stationary from the perspective of any individual agent as each agent's policy changes during training. The non-stationarity is a result of evolving policies which prevent any one agent from predicting the next state based on their individual actions.

When multiple agents have goals that are interdependent on the actions performed by each other, as in the case of hierarchical environments, it becomes very challenging to learn robust policies. Recent works, such as counterfactual multi-agent policy gradients [8] and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [24], have used the centralized training of decentralized policies paradigm to overcome the non-stationarity issue in multi-agent training. However, these methods do not consider hierarchies that exist among agents in certain domains.

In this work, we focus on integrating the Stackelberg equilibrium concept into multi-agent reinforcement learning to learn policies that take advantage of positions within an

hierarchy for domains with continuous state and action spaces. We propose a new algorithm, Stackelberg Multi-Agent Reinforcement Learning (SMARL), which is an extension of the MADDPG algorithm [24] for hierarchical environments with asymmetric information. The proposed algorithm requires formulating a new Stackelberg learning objective, which conditions policies on actions of other agents. This in-turn requires a new method for optimizing the Stackelberg learning objective for multi-agent continuous actions. Due to the dynamics explored in the proposed SMARL algorithm, there are a lack of environments available to test the proposed changes. This requires implementation of a new environment that is also compatible with existing algorithms. The use of new and modified environments allows for evaluation of our proposed changes and comparison to baselines, on multiple mixed cooperative and competitive domains.

1.3 Summary of Contributions

This thesis aims to understand how hierarchical structures within environments, whether they are naturally occurring or artificially induced, can lead to better methods for training policies in multi-agent reinforcement learning.

- In chapter 2, we summarize some related works upon which we build our proposed algorithm, along with background concepts required to understand the literature.
  - We introduce the fundamental concepts in reinforcement learning such as Markov decision processes, Q-learning, policy gradient and deterministic policy gradient algorithms in section 2.1.
  - Multi-agent actor-critic methods, which act as the base upon which we implement our changes are discussed in section 2.2.
  - In section 2.3, we present prior works which have applied the Stackelberg equilibrium model to MARL and discuss how they differ from our contributions.
- In chapter 3, we apply the Stackelberg equilibrium model to multi-agent reinforcement learning domains with continuous action spaces.
  - We formulate the Stackelberg learning objective in section 3.1. This includes defining the concept of leaders and followers for multi-level hierarchies and integrating this idea into a solvable objective function.

  - In section 3.2, we detail the actor-critic policy updates using our Stackelberg learning objective. We show the use of follower responses, resulting from leader enforcement, in policy gradient and temporal difference methods for updating actor and critic policies respectively.

- In chapter 4, we study the relation between total number of agents and stability of learned policies.
  - We discuss the problem of having multiple followers in section 4.1. We show how increased followers add noise to the gradient of our Stackelberg learning objective function.
  - In section 4.2, we propose a method to reduce the noise in gradient updates, caused by multiple followers, by normalizing follower contributions to the policy gradient update.
  - In section 4.3, we explore the use of schedules to vary the magnitude of follower influence on gradient updates throughout training. This is done to improve policy robustness and time to convergence.

- In chapter 5, we present experimental domains used to test the performance of our algorithm, along with the results obtained from these experiments.
  - We detail our custom highway driving environment and the modifications made to two multi-agent particle environments in section 5.1. We also discuss the observation and action spaces along with reward structures for all agents.
  - In section 5.2, we present the results obtained and explain their significance with comparisons to a baseline algorithm.

- In chapter 6, we conclude the thesis and provide actionable items for future directions.

Chapter 2

Related Work and Background

This chapter provides an overview of the different sets of literature related to game theory and multi-agent reinforcement learning, along with key algorithms existing methods and our own are built upon. We focus on the algorithms used for training policies in single-agent and multi-agent mixed cooperative and competitive settings, the Stackelberg equilibrium concept and formulating it as a multi-agent reinforcement learning problem, and the use of game theory in autonomous driving domains. We also comment on how the literature relates to and differs from our work.

2.1 Reinforcement Learning Methods

In the following section we focus on the topics in reinforcement learning that are relevant to understanding our contributions in formulating the Stackelberg equilibrium concept for continuous multi-agent reinforcement learning with asymmetric information and structure.

2.1.1 Markov Decision Process and Markov Games

The standard reinforcement learning problems where the outcomes are a combination of random chance and active choices can be analyzed and optimized using a Markov Decision Process (MDP). We define a MDP with a state space $\mathcal{S}$, real-valued action space $\mathcal{A} = \mathbb{R}^N$, initial state distribution $p(s_1)$, transition dynamics $p(s_{t+1}|s_t, a_t)$ and reward function $r(a_t, a_t)$. A single-agent reinforcement learning setup contains an agent interacting with an environment E in discrete time, where at each time step t the agent receives an observation

o_t, takes action $a_t \in \mathcal{A}$ and receives reward r_t. A policy π is used to map states to a probability distribution over actions $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$ and is representative of the agent's behaviour [21]. The return from a state at time t is defined as the sum of discounted future rewards $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$, with time horizon T and a discount factor $\gamma \in [0, 1]$.

In multi-agent reinforcement learning, an extension of Markov decision processes known as partially observable Markov games [23] are used to reason about multi-agent environments. A Markov game consisting of N agents is defined by a set of states $\mathcal{S}$ describing the possible configurations of all agents, a space of actions $\mathcal{A}_1, \ldots, \mathcal{A}_N$, and a set of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$ for each agent. Each agent i selects its action using a stochastic policy $\boldsymbol{\pi}_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$, parameterized by θ_i. These actions produce the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \mapsto \mathcal{S}$, while each agent receives a private observation correlated with the state $\mathbf{o}_i : \mathcal{S} \mapsto \mathcal{O}_i$. The reward for each agent is a function of the state and the actions taken by all agents $r_i : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \mapsto \mathbb{R}$. Initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$. Each agent i wants to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$ with discount factor γ and time horizon T.

2.1.2 Q-Learning and Deep Q-Networks (DQN)

Q-learning [38] is an off-policy model-free reinforcement learning algorithm which uses an action-value (Q) function for policy π as $Q^\pi(s, a) = E[R|s^t = s, a^t = a]$, to learn the optimal policy without learning a model of the environment. The Q function is a measure of the overall expected reward given that the agent is in state s^t performing action a^t and the optimal action-value function is given by $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. This Q function can be written recursively as what is called the Bellman equation [3]:

$$Q^\pi(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^\pi(s', a')]], \tag{2.1}$$

where s' is the new state reached after taking action a in state s, while a' is the action taken in state s'. This method is widely used in single agent games and has also been previously applied to multi-agent stochastic games [36] where agents choose the best mixed strategy given the expected mixed strategy of all other agents.

In single-agent Deep Q-Networks (DQN) [25], the action-value function is represented using a neural network parameterized by θ. It learns the optimal function Q^* corresponding to the optimal policy by minimizing the loss:

$$\begin{aligned} L(\theta) &= \mathbb{E}_{s,a,r,s'}[(Q^*(s, a|\theta) - y)^2], \\ \textit{where} \quad y &= r(s, a) + \gamma \max_{a'} \bar{Q}^*(s', a'), \end{aligned} \tag{2.2}$$

where the target values y are computed using the target action-value function $\bar{Q}$. These two Q functions have the same network structure, but the parameters of $\bar{Q}$ are updated periodically with the most recent θ corresponding to the optimal policy, which helps stabilize learning. DQN also leverages an experience replay buffer D containing tuples (s, a, r, s') to stabilize learning by making more efficient use of previous experience. Experience replay helps separate learning from experience gathering, since the learning is not restricted to state-action pairs generated by the current policy. This is important because the limited samples produced by a policy at an instance in time contain low variations in its states and actions. The experience replay buffer helps preserve past experience containing large variations in states and actions, enabling efficient use of samples for learning.

Q-learning can be applied to a multi-agent setting by having each agent i learn an independent optimal function Q_i [35]. However, as learning progresses the agents are independently updating their policies which results in a non-stationary environment from the view point of any one agent. This violates the Markov assumptions required for the convergence of Q-learning. This non-stationarity issue is addressed by the centralized training and decentralized execution framework discussed in section 2.2 below.

2.1.3 Policy Gradient (PG) Algorithms

The objective of reinforcement learning is to find the optimal behaviour strategy for an agent to maximize its expected return. Policy gradient methods address this by modelling and directly optimizing a policy π parameterized by θ. The policy is optimized by maximizing the objective function which depends on the policy, this objective function is defined as:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q^{\pi}(s, a), \tag{2.3}$$

where $d^{\pi}(s)$ is the on-policy state distribution under π. Various algorithms can be used to maximize the objective by directly adjusting parameters θ of policy π by taking steps in the direction of the gradient $\nabla_{\theta} J(\theta)$. Using the policy gradient theorem [34] the gradient of the objective function is written as:

$$\begin{aligned} \nabla_{\theta} J(\theta) &\propto \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) \\ &= \mathbb{E}_{s \sim d^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)]. \end{aligned} \tag{2.4}$$

This is useful in continuous space environments where there are an infinite number of action and states, which make it computationally intractable to maximize over the action

space as done in section 2.1.2. Continuous policy parameterization also allows for smooth changes to action probabilities through learned parameters, whereas in ϵ-greedy the action probabilities may change dramatically for small changes in estimated action values [34].

There are multiple methods of estimating Q^π resulting in different algorithms. One simple estimate is using the sample return $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$, which leads to the REINFORCE algorithm [39]. A variation of REINFORCE if to subtract a baseline value from Q^π to reduces the variance of gradient estimation while keeping the bias unchanged [29]. Reducing the variance is important since policy gradient methods are known to exhibit high variance gradient estimates, which are made worse in multi-agent settings since an agent's reward depend on the actions of multiple agents. If the optimization process does not consider the actions of other agents it leads to more variable returns, thereby increasing the variance of its gradients. These algorithms are known as *model-free* algorithms because they can learn a policy directly using policy gradients and do not require a representation of their environment. *Model-based* algorithms try to understand the environment by creating models which capture state transitions and the reward function.

Another approach for estimating Q^π is approximating the true action-value function $Q^\pi(s, a)$ known as the *critic* which leads to a class of actor-critic algorithms [33]. These will be the focus of the next few sections and integral to our contributions.

2.1.4 Deterministic Policy Gradient (DPG) Algorithms

The methods described in previous sections have all used stochastic policy functions $\pi(.|s)$ which model the probability distribution over actions $\mathcal{A}$ given the current state. Deterministic policy gradient instead models policies as deterministic decisions $a = \mu_\theta(s) : \mathcal{S} \mapsto \mathcal{A}$. Given the discounted state distribution $\rho^\mu(s)$ the performance objective can be written as an expectation:

$$\begin{aligned} J(\theta) &= \int_{\mathcal{S}} \rho^\mu(s) r(s, \mu_\theta(s)) \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\mu}[r(s, \mu_\theta(s))] \end{aligned} \tag{2.5}$$

The gradient of the objective function $J(\theta)$ is computed using the *Deterministic Policy Gradient Theorem* [31] and applying the chain rule as follows:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}], \end{aligned} \tag{2.6}$$

where the action-value function $Q^{\mu}(s, a)$ is used to estimate the return. Since $\nabla_{\theta} J(\theta)$ relies on $\nabla_a Q^{\mu}(s, a)$, it is required that the action space $\mathcal{A}$ and consequently the policy μ_{θ} need to be continuous. The deterministic policy gradient is a special case of the stochastic policy gradient where the probability distribution contains one extreme non-zero value over one action. More concretely, [31] shows that if stochastic policies $\pi_{\mu_{\theta},\sigma}$ are parameterized by a deterministic policy μ_{θ} and a variance parameter σ, then for $\sigma = 0$ the stochastic policy is equivalent to the deterministic policy $\pi_{\mu_{\theta},0} \equiv \mu_{\theta}$. As $\sigma \to 0$ the stochastic policy gradient converges to the deterministic gradient.

Deep deterministic policy gradient (DDPG) [21] is a model-free, off-policy actor-critic algorithm that extends DPG by using deep neural networks to approximate the policy μ and critic Q^{μ}. DDPG also builds on the work done in DQN (Deep Q-Network) which uses an experience replay buffer and target networks to stabilize Q-learning, but unlike DQN which freezes the target networks for some period of time, DDPG does soft updates which update the target networks slowly but consistently. This is done by updating the parameters of both the actor and critic using $\tau \ll 1 : \theta' \leftarrow \tau\theta + (1-\tau)\theta'$. While the original DQN worked with discrete space, DDPG extends it to continuous space while learning a deterministic policy. The use of a deterministic policy is not good for exploration; to address this DDPG uses an exploration policy μ' that is constructed by adding noise $\mathcal{N}$ to the deterministic policy: $\mu'(s) = \mu_{\theta}(s) + \mathcal{N}$. The next section looks at applying these algorithms to multi-agent domains.

2.2 Multi-Agent Actor-Critic Methods

We look at the actor-critic [14] formulation from the perspective of policy gradient methods where the actor represents the policy and critic represents the value function. Value functions help guide policy updates by reducing gradient variance seen in the vanilla policy gradient method discussed in section 2.1.3. The critic can be represented as either an action-value function $Q_{\omega}(a|s)$ or a state-value function $V_{\omega}(s)$ with parameters ω. The actor can be represented as either a stochastic $\pi_{\theta}(a|s)$ or deterministic $\mu_{\theta}(s)$ policy with parameters θ. Critics use temporal difference (TD) learning with a linear approximation architecture [14] to perform updates, while the actors follow the gradient computed using information provided by the critic. Learning rates, α_{θ} and α_{ω}, are predefined for actor and critic parameter updates respectively.

The actor-critic framework forms the basis of many multi-agent reinforcement learning algorithms that we study and build our algorithm upon. In the following sections we discuss current state of the art multi-agent algorithms that helped shape our contributions. This

section is divided into three parts which discuss multi-agent actor-critic methods, methods for communication between agents and influencing the policies of other agents.

2.2.1 Multi-Agent Deep Deterministic Policy Gradient

As stated previously, single-agent RL algorithms cannot be directly applied to a multi-agent setting since they violate the Markov assumptions required for convergence. A main factor being the non-stationarity of the environment from the viewpoint of any one agent. The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm [24] addresses this issue by having each agent learn a centralized Q function which conditions on the global information to mitigate the non-stationarity problem and stabilize training.

Consider a game consisting of N agents with continuous policies $\boldsymbol{\mu}_{\theta_i}$ (abbreviated as $\boldsymbol{\mu}_i$) parameterized by $\boldsymbol{\theta} = \{\theta_1, ..., \theta_N\}$, and a set of all agent policies $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_N\}$. Then the gradient of the expected return for each agent i with policy $\boldsymbol{\mu}_i$, $J(\theta_i) = \mathbb{E}[R_i]$, can be written as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x}, a \sim D}[\nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)|_{a_i = \boldsymbol{\mu}_i(o_i)}]. \tag{2.7}$$

In this case, the $Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)$ is a *centralized action-value function* which takes as input the actions of all agents, $a_1, ..., a_N$, along with some state information $\mathbf{x}$, and outputs the Q-value for each agent i. The state information $\mathbf{x}$ could contain the observations of all agents, $\mathbf{x} = (o_1, ..., o_N)$, including additional state information if available. The experience replay buffer D contains tuples $(\mathbf{x}, \mathbf{x}', a_1, ..., a_N, r_1, ..., r_N)$, where $\mathbf{x}'$ denotes the next state from $\mathbf{x}$ after taking action $a_1, ..., a_N$. The centralized action-value function $Q_i^{\boldsymbol{\mu}}$ is updated by minimizing the loss function defined as:

$$\begin{aligned} \mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N) - y)^2], \\ y &= r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a'_1, ..., a'_N)|_{a'_j = \mu'_j(o_j)}, \end{aligned} \tag{2.8}$$

where $\boldsymbol{\mu}' = \left\{\boldsymbol{\mu}_{\theta'_1}, ..., \boldsymbol{\mu}_{\theta'_N}\right\}$ is the set of target policies with delayed parameters θ'_i. Since the action-value function for each agent $Q_i^{\boldsymbol{\mu}}$ is learned separately, agents can have arbitrary reward structures, including conflicting rewards in competitive settings.

The centralized Q function is only required during training. While during execution, each agent's policy $\boldsymbol{\mu}_{\theta_i}$ only takes as input the local information o_i to produce an action. This method is presented below in Algorithm 1 since many other methods including our contributions are built upon this centralized critic decentralized actor framework.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents [24]

for *episode* $= 1$ *to* M **do**

  Initialize a random process $\mathcal{N}$ for action exploration

  Receive initial state $\mathbf{x}$

  for $t = 1$ *to max-episode-length* **do**

    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration

    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}$'

    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$

    $\mathbf{x} \leftarrow \mathbf{x}'$

    for *agent* $i = 1$ *to* N **do**

      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$

      Set $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$

      Update critic by minimizing the loss

$$\mathcal{L}(\theta_i) = \frac{1}{S} \sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j) \right)^2$$

      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end

    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$$

  end

end

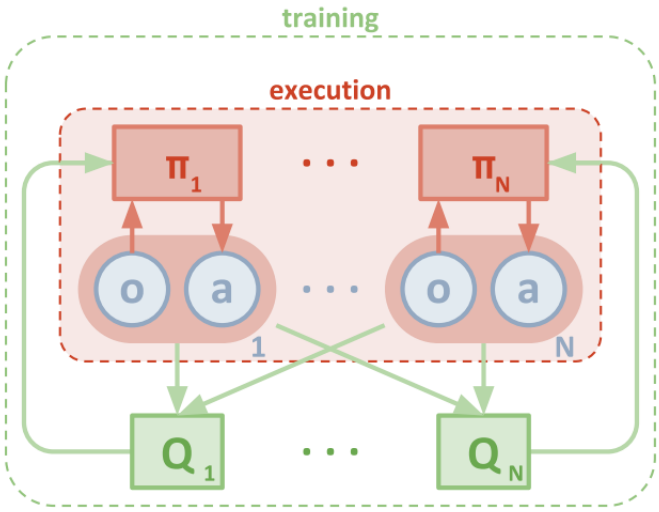

Figure 2.1: Overview of multi-agent decentralized actor, centralized critic approach [24].

2.2.2 Minimax Multi-Agent Deep Deterministic Policy Gradient

The Minimax Multi-Agent Deep Deterministic Policy Gradient (M3DDPG) [20] is an extension of MADDPG algorithm and is designed to improve the robustness of learned policies. This is done by introducing the minimax optimization from game theory into the learning objective. M3DDPG proposes to update policies considering the worst situation during training, by optimizing the accumulative reward for each agent i under the assumption that all other agents act adversarially. This results in a minimax learning objective $\max_{\theta_i} J_M(\theta_i)$ where:

$$\begin{aligned} J_M(\theta_i) &= \mathbb{E}_{s\sim\rho^\mu}[R_i] \\ &= \min_{a^t_{j\neq i}} \mathbb{E}_{s\sim\rho^\mu}\Big[\sum_{t=0}^{T} \gamma^t r_i(s^t, a^t_1, \ldots, a^t_N) \mid_{a^t_i=\mu(o^t_i)}\Big] \\ &= \mathbb{E}_{s^0\sim\rho}\Big[\min_{a^t_{j\neq i}} Q^\mu_{M,i}(s^0, a^0_1, \ldots, a^0_N) \mid_{a^t_i=\mu(o^t_i)}\Big]. \end{aligned}$$

From this minimax learning objective we can see that for any agent i, none of the adversarial actions depend on its parameters θ_i. This allows for the direct application of the deterministic policy gradient theorem to compute $\nabla_{\theta_i} J_M(\theta_i)$ and off-policy temporal difference to update the Q function. The minimization term in the above equation poses

an issue since optimizing the minimax objective becomes computationally intractable in continuous action spaces. M3DDPG addresses this optimization problem by approximating the non-linear Q functions by a locally linear function and replacing the inner-loop minimization with single step gradient descent.

In order to approximate the non-linear Q function, M3DDPG interprets the minimization of the Q function as seeking a set of perturbations ϵ, such that the perturbed actions $a'^{*} = a' + \epsilon$ decrease the Q value the most. Linearizing the Q function at $Q^{\mu}_{M,i}(x, a'_1, \ldots, a'_N)$, allows these perturbations to be locally approximated by the gradient direction at $Q^{\mu}_{M,i}(x, a'_1, \ldots, a'_N)$ w.r.t a'_j:

$$\forall j \neq i,\ \hat{\epsilon}_j = -\alpha \nabla_{a_j} Q^{\mu'}_{M,i}(x', a'_1, \ldots, a_j, \ldots, a'_N),$$

where α is a hyper-parameter representing the perturbation rate. Using these perturbations, the gradient of the minimax objective is written as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x}, a \sim D} \begin{bmatrix} \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{M,i}(\mathbf{x}, a^*_1, \ldots, a_i, \ldots, a^*_N) | \\ a_i = \boldsymbol{\mu}_i(o_i) \\ a^*_j = a_j + \hat{\epsilon}_j,\ \forall j \neq i \\ \hat{\epsilon}_j = -\alpha_j \nabla_{a_j} Q^{\mu}_{M,i}(x, a_1, \ldots, a_N) \end{bmatrix} \tag{2.9}$$

The action-value function $Q^{\mu}_{M,i}$ is updated using temporal difference as done in Eq. 2.8 with the addition of adversarial actions, resulting in a loss function whose parameters are updated by minimizing the error defined as:

$$\begin{aligned} \mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'}[(Q^{\boldsymbol{\mu}}_{M,i}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \\ y &= r_i + \gamma Q^{\boldsymbol{\mu}'}_{M,i}(\mathbf{x}', a'^{*}_1, \ldots, a'_i, \ldots, a'^{*}_N) \\ a'_k &= \mu'_k(o_k),\ \forall 1 \leq k \leq N \\ a'^{*}_j &= a'_j + \hat{\epsilon}'_j,\ \forall j \neq i \\ \hat{\epsilon}'_j &= -\alpha_j \nabla_{a'_j} Q^{\mu}_{M,i}(x, a'_1, \ldots, a'_N) \end{aligned} \tag{2.10}$$

This method is relevant to our contributions since it shows that it is possible to apply a game theoretic concept to multi-agent reinforcement learning with continuous action space.

2.2.3 Counterfactual Multi-Agent Policy Gradients

Counterfactual multi-agent (COMA) policy gradients [8] is another method for training policies in a multi-agent setting using a centralized critic and decentralized actors. This

method is more focused on cooperative settings as opposed to the competitive settings in section 2.2.2 and the mixed cooperative and competitive settings in 2.2.1. In addition to learning policies, this method addresses the multi-agent credit assignment problem [4] encountered in cooperative settings due to the shared global rewards, which makes it difficult for each agent to deduce its contributions to the overall goal.

COMA is based on three main ideas. The first idea is the use of a single centralized critic as opposed to multiple centralized critics presented in previous sections. The second idea is the use of counterfactual baselines, inspired by difference rewards [40, 37]. Third is the use of critic representation that allows counterfactual baselines to be computed efficiently. We focus on the second idea, as it is most relevant to our work.

Difference rewards compare the global reward to the reward received when an agent's action is replaced by a default action. In a similar theme, COMA uses its centralized critic to compute an agent specific advantage function which compares the estimated return from the current joint actions to a counterfactual baseline that marginalizes out a single agent's action while keeping the other agents' actions fixed.

The centralized critic is used to implement these difference rewards. This centralized critic is represented as $Q(s, \mathbf{u})$, for joint action $\mathbf{u}$ conditioned on global state s; its the same as the previous sections but with different notation. The difference reward is represented by the advantage function $A^a(s, \mathbf{u})$ which compares Q-values for current action u^a to a counterfactual baseline that marginalizes out u^a, while keeping the other agents' actions $\mathbf{u}^{-a}$ fixed. This advantage function is defined as:

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a)|\tau^a)Q(s, (\mathbf{u}^{-a}, u'^a)). \tag{2.11}$$

Though the counterfactual baseline is dependent on the policy, its expected contribution to the gradient is zero [8]; hence it does not introduce bias. Given this fact, the COMA gradient is given by:

$$\begin{aligned} g &= \mathbb{E}_\pi \Bigg[\sum_a \nabla_\theta \log \pi^a(u^a \mid \tau^a) A^a(s, \mathbf{u}) \Bigg] \\ &= \mathbb{E}_\pi \Bigg[\sum_a \nabla_\theta \log \pi^a(u^a \mid \tau^a) Q(s, \mathbf{u}) \Bigg] \end{aligned} \tag{2.12}$$

COMA uses the counterfactual baseline to address the multi-agent credit assignment problem in cooperative environments. We show in Section 2.3 that the process of computing

the counterfactual baseline is comparable to computing the Stackelberg actions in discrete action spaces and provide the basis for extending the Stackelberg concept to continuous action spaces.

2.3 Stackelberg Model

The Stackelberg equilibrium model is a concept from game theory used to model duopolies. While there are other methods for modelling duopolies such as the Cournot and Bertrand models [2], Stackelberg is unique in that it allows one player to commit to an action before the other player. The player that is allowed to move first is called a leader and the other agent is labeled a follower. An important assumption made in these models is that each player knows the other players reaction function. This means that the leader knows how the follower will react to its action, so it picks an action that maximizes its payoff using the follower's reaction function. Another point to note is that once the leader commits to an action, it cannot be changed. After the leader has *committed* to an action, the follower reacts to maximize its payoff under the strategy imposed on it by the leader. Simultaneous execution of actions is allowed as long as the leader committed to their action before the follower. This setup of leader and follower is the Stackelberg model/game and actions resulting from solving this model is the Stackelberg equilibrium. To illustrate this model, consider the following one iteration game with continuous actions.

Stackelberg Game Example:

Players: Firms Apex and Brydox
Order of Play:

1. Apex chooses quantity q_a from the set $[0; \infty)$
2. Brydox chooses quantity q_b from the set $[0; \infty)$

Payoffs:

Payoffs π are profits given by price times quantity minus cost of production.

- Cost: $c = 12$
- Total quantity: $Q = q_a + q_b$
- Price: $p(Q) = 120 - q_a - q_b$

$$\pi_{Apex} = (120 - q_a - q_b)q_a - cq_a = (120 - c)q_a - q_a^2 - q_a q_b$$
$$\pi_{Brydox} = (120 - q_a - q_b)q_b - cq_b = (120 - c)q_b - q_b^2 - q_a q_b$$

Brydox's best response resulting from maximizing π_{Brydox} w.r.t q_b:

$$q_b = 60 - \frac{q_a + c}{2}$$

Apex substitutes Brydox's reactions function into its payoff to obtain:

$$\pi_{Apex} = (120 - c)q_a - q_a^2 - q_a(60 - \frac{q_a + c}{2})$$

Maximizing π_{Apex} w.r.t q_a yields $q_a = 54$ and consequently $q_b = 27$. The Stackelberg equilibrium resulting from solving this game is the strategy $(q_a = 54, q_b = 27)$.

This model has been extended to multi-agent reinforcement learning settings to solve various problems. In the following section we look at some works that have applied the Stackelberg model to Q-learning and policy gradient methods and discuss how our contributions are different.

2.3.1 Asymmetric Multi-Agent Reinforcement Learning

The work titled Asymmetric Multiagent Reinforcement Learning [15] applied the Stackelberg model to multi-agent Markov games with asymmetric information. In particular, two agent matrix games are analyzed and Q-learning is used to learn policies for each agent. In a Stackelberg game, asymmetry occurs when one agent has the ability to enforce their strategy onto other agents. This results in a hierarchy where one agent acts as the leader and the others as followers. Consider a two player finite game with discrete actions, where player 1 is the leader and player 2 is the follower. The Stackelberg solution $(a_S^1, a_S^2(a^1))$ is modeled by the following two equations:

$$a_S^1 = \text{argmax}_{a^1 \in A^1} r^1(a^1, a_S^2(a^1))$$
$$a_S^2(a^1) = \text{argmax}_{a^2 \in A^2} r^2(a^1, a^2)$$

where a_S^i is the action corresponding to agent i, satisfying the Stackelberg equilibrium. The follower's response is expressed as a function of the leader's action since the leader has the ability to enforce its strategy onto the follower to elicit a more desireable outcome.

The work in [15] extends the Stackelberg model to multi-agent reinforcement learning by representing agent strategies using policies and using Q-learning for training. More

concretely, given a two player game with finite policies π^1 and π^2, the leader policy takes as input the current state and produces a distribution over actions, $\pi^1 : S \mapsto A^1$, while the follower policy takes as input the current state and leader action, $\pi^2 : S \times A^1 \mapsto A^2$. The action-value function $Q^i_{\pi^1,\pi^2}(s, a^1, a^2) = E_{\pi^1,\pi^2}[R^i | s_t = s, a^1_t = a^1, a^2_t = a^2]$ is updated using three stages that solve a Stackelberg equilibrium solution for a state $s \in S$.

Stages:

1. Determine cooperation strategy $a^c = (a^{1c}, a^{2c})$ that maximizes $Q^1_{\pi^1,\pi^2}$ in state s.

$$\arg \max_{\substack{a^1 \in A^1 \\ a^2 \in A^2}} Q^1_{\pi^1,\pi^2}(s, a^1, a^2) \tag{2.13}$$

2. Determine leader's enforcement action $a^1_S = g(s, a^c)$:

$$g(s, a^c) = \arg \min_{a^1 \in A^1} \parallel \arg \max_{a^2 \in A^2} (Q^2_{\pi^1,\pi^2}(s, a^1, a^2)), a^c \parallel, \tag{2.14}$$

$$\text{where } \parallel x, a^c \parallel = |Q^1_{\pi^1,\pi^2}(s, a^1, x)) - Q^1_{\pi^1,\pi^2}(s, a^{1c}, a^{2c}))|.$$

3. Determine follower's response a^2_S

$$a^2_S = \arg \max_{a^2 \in A^2} Q^2_{\pi^1,\pi^2}(s, g(s, a^c), a^2) \tag{2.15}$$

In Eq. 2.14 above, $g(s, a^c)$ is simply selecting an action a^1_S such that player 2's response a^2_S is close to the cooperative strategy a^c, which benefits the leader. The resulting strategy (a^1_S, a^2_S) is in a Stackelberg equilibrium and used to update the Q-functions for the leader and follower:

$$\begin{aligned} Q^1_{t+1}(s_t, a^1_t, a^2_t) = {} & (1 - \alpha_t) Q^1_t(s_t, a^1_t, a^2_t) \\ & + \alpha_t [r^1_{t+1} + \gamma \max_{a^1_{t+1} \in A^1} Q^1_t(s_{t+1}, a^1_{t+1}, \pi^2(s_{t+1}, a^1_{t+1}))], \end{aligned} \tag{2.16}$$

$$\begin{aligned} Q^2_{t+1}(s_t, a^1_t, a^2_t) = {} & (1 - \alpha_t) Q^2_t(s_t, a^1_t, a^2_t) \\ & + \alpha_t [r^2_{t+1} + \gamma \max_{a^2_{t+1} \in A^2} Q^2_t(s_{t+1}, g(s_{t+1}, a^c_{t+1}), a^2_{t+1})] \end{aligned} \tag{2.17}$$

where r_t is the immediate reward at time step t, γ is the discount factor and α is the learning rate. Note that only the leader needs a copy of the follower's Q function to compute its action $a^1_S = g(s_t, a^c)$, which is enforced onto the follower.

The method proposed in [15] solves a Stackelberg game to determine the best return in state s_{t+1} resulting from taking actions (a_t^1, a_t^2) in state s, instead of simply taking the greedy action as done in standard Q-learning. This is different from our work where we are applying the Stackelberg model to continuous action spaces, which prevents us from maximizing over a set of discrete actions. Our work also extends to environments with more than two agents and leverages policy gradient methods to train policies. Though our work differs from [15], we take into consideration the Q function update rule, which modifies the target values to consider the Stackelberg equilibrium strategies.

2.3.2 Gradient Methods for Stackelberg Security Games

A gradient based approach for solving Stackelberg games in security settings is presented in [1]. The game contains two players, where the leader is the defender who must allocate resources $R = \{r_1, \ldots, r_K\}$, to protect targets $T = \{t_1, \ldots, t_N\}$, while the attacker (follower) has to determine which subset of the targets to attack. A defender's pure strategy is an assignment of resources to schedules $s \in \mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_K$, where a schedule s_k represents the targets to which resource r_k is assigned. An attacker's pure strategy is simply a target $t_n \in T$ to attack. A pure strategy is a set of moves that the player plays with certainty throughout the game. There can be multiple pure strategies, but the player uses only one throughout the game. In contrast, the defender plays a mixed strategy which is a distribution $\mathcal{D}$ over pure strategies. A set of the defender's mixed strategies are denoted Δ, and an attacker's response function g maps a mixed strategy to a set of targets $g : \Delta \to T$.

In order to solve the Stackelberg model using gradient methods, this work [1] assumes a probabilistic model for the attacker and restricts the set of strategies available to the defender. This probabilistic model is a categorical distribution over the set of targets T, which selects an element i with probability $\sigma_\eta(\mathbf{x}, i)$ the softmax distribution, with some vector $\mathbf{x} \in \mathbb{R}^N$. The attacker selects its target according to a softmax distribution over its utilities, where utility function $\mathbf{U}_\lambda$, corresponding to attacker type λ, generates the payoff given a mixed strategy $\mathcal{D}$ and some target t. $\mathbf{u}_\lambda(\mathcal{D})$ is a vector of expected attacker payoffs for each choice of target $t \in T$. The attacker selects target t_n with probability $\sigma_\eta(\mathbf{u}_\lambda(\mathcal{D}), n)$.

The defender is restricted to select distributions from a parameterized class $\Delta_\Theta \subset \Delta$, where Θ is a set of parameters, and for each $\theta \in \Theta$, there is a corresponding distribution $\mathcal{D}_\theta \in \Delta_\Theta$. The probability mass function of $\mathcal{D}_\theta$ over the schedules $\mathcal{S}$ is denoted by $p(\cdot|\theta)$. These approximations produce a stochastic optimization problem from the perspective of

the defender. The goal of the defender is to maximize its expected payoff given by:

$$\tilde{\mathbf{U}}_d(\theta) = \mathbf{E}\big[U_d(\mathbf{c}_n(\mathbf{s}), t_n)\big], \tag{2.18}$$

where t_n is the target selected by the attacker according to $\sigma_\eta(\mathbf{u}_\lambda(\mathcal{D}_\theta), \cdot)$, and $\mathbf{c}_n(\mathbf{s}$ is the coverage of target n under schedule $\mathbf{s}$, where $\mathbf{s} \sim \mathcal{D}_\theta$.

We do not focus on the solution to this formulation since it is not relevant to our contributions. The main takeaway from the work in [1] is the use of a probabilistic model for the attacker which conditions on the parameterized distribution $\mathcal{D}_\theta$ of the defender, which allows the defender to update its parameters θ using gradient ascent while taking the attacker response into account. Our work uses this idea of having followers condition of the leader parameters, but it differs from this work as our system is formulated as a multi-agent reinforcement learning problem.

2.4 Summary

This chapter looked at some of the fundamental algorithms in reinforcement learning used to train policies in Section 2.1, multi-agent reinforcement learning methods that train multiple agents in cooperative and competitive settings in Section 2.2 and works that applied reinforcement learning methods to represent and solve Stackelberg models in Section 2.3. Works that have extended the Stackelberg model to be solved using reinforcement learning methods have limited the applicability of the model to two agent discrete action domains. Our work aims to extend the Stackelberg model to any number of agents with continuous action spaces. This requires us to study and build on the work done in multi-agent reinforcement learning. We use the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm, detailed in section 2.2.1, as our base upon which we implement our changes. Work in M3DDPG from section 2.2.2, the minimax version of MADDPG, demonstrated a method for approximating minimization over a continuous action space and scaling the gradient to control noise. This is useful to our work as it allows us to control the impact of follower responses on leader policy updates. The counterfactual baseline method in section 2.2.3 allows the Q-learning Stackelberg formulation from section 2.3.1 to generalize to a group of agents. In our work we extend the Stackelberg models formulated for discrete action spaces to multi-agent reinforcement learning with multiple hierarchies, agents and continuous action spaces.

Chapter 3

Stackelberg Multi-Agent Reinforcement Learning

In this chapter, we focus on the problem of applying the Stackelberg equilibrium model to multi-agent reinforcement learning domains with continuous action spaces. Previous works have extended the Stackelberg model to domains with a restricted number of agents and discrete action spaces. We present a general formulation which can be applied to any number of agents operating in continuous domains with mixed cooperative and competitive objectives. Our new algorithm used to solve this formulation is called *Stackelberg Multi-Agent Reinforcement Learning (SMARL)*. The following sections present our new Stackelberg learning objective which takes into account influence from multiple followers as well as the method for maximizing this objective, since existing solutions are intractable for continuous action spaces.

3.1 Stackelberg Learning Objective

In many multi-agent domains, there are hierarchies within the environment which give some agents an advantage over the others. Agents at the top of the hierarchy are known as leaders and have the ability to commit to an action before other agents. Agents lower in the hierarchy are known as followers and observe the actions selected by the leaders. The Stackelberg model discussed in section 2.3 assumes that all agents have knowledge of other agent reaction functions. Given this knowledge of reaction functions and the ability of leaders to commit their actions before followers results in asymmetry within the

environment. In order to account for these asymmetries and learn policies for leaders and followers, we employ the Stackelberg equilibrium formulation.

3.1.1 Multi-Level Hierarchy

The strategies and reaction functions in multi-agent reinforcement learning (MARL) settings are represented by the agent's policy. These policies $\boldsymbol{\mu}$ are function approximators represented by neural networks whose weights are parameterized by θ. During training, we optimize the expected return for the leaders by having them condition on follower policies. The leaders' actions at each step are communicated to the followers and become part of their observations. The process of communicating leader actions to the followers enables the followers to observe the leader commitments. This communication mechanism is illustrated using Figure 3.1, which contains three levels of hierarchy from highest to lowest starting at the top. The observations for each agent are as follows:

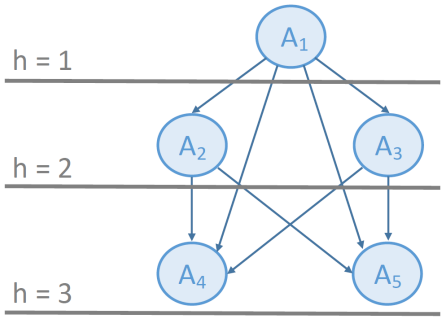

Figure 3.1: Three levels of hierarchy where followers observe leaders' actions

$$
\begin{aligned}
a_1 &= \boldsymbol{\mu}_1(o_1) \\
a_2 &= \boldsymbol{\mu}_2(o_2, a_1), \;\; a_3 = \boldsymbol{\mu}_3(o_3, a_1) \\
a_4 &= \boldsymbol{\mu}_4(o_4, a_1, a_2, a_3), \;\; a_5 = \boldsymbol{\mu}_5(o_5, a_1, a_2, a_3)
\end{aligned}
$$

where $a_i, \boldsymbol{\mu}_i, o_i$ are the action, deterministic policy and private observation, respectively, for agent i. Agent 1 is at the top of the hierarchy and does not have any leader; hence, its policy only takes as input its private observation. Agents 2 and 3 observe agent 1's action in addition to their private observations, while agents 4 and 5 observe actions a_1, a_2 and a_3. We use deterministic policies since they allow us to apply policy gradient methods for

continuous action values as discussed in section 2.1.4. We can generalize this notation for any number of hierarchies H, where $\hat{H} = \{1, \ldots, H\}$ is a set of hierarchy levels, and each level $h \in \hat{H}$ contains some non-zero set of agents.

In order to generalize the above notation, consider a set of N agents, $\hat{N} = \{1, \ldots, N\}$, where each value in the set $\hat{N}$ corresponds to an agent id. Let $\mathcal{F}(i) \subseteq \hat{N}$ represent the set of followers and $\mathcal{L}(i) \subseteq \hat{N}$ the set of leaders for agent i. The set of agents at level h in the hierarchy is given by $\mathcal{H}(h)$. Assuming agent i is at some hierarchy level h, then its follower set contains all agents from lower hierarchy levels, which corresponds to a higher h value, $\mathcal{F}(i) = \{\mathcal{H}(h+1), \ldots, \mathcal{H}(H)\}$. Similarly, the leader set contains all agents from higher levels, $\mathcal{L}(i) = \{\mathcal{H}(1), \ldots, \mathcal{H}(h-1)\}$. The intersection of leader, follower sets for any agent i is an empty set $\mathcal{F}(i) \cap \mathcal{L}(i) = \emptyset$, since all other agents can only occupy one of H hierarchy levels at any given time. This means that each agent in the set $\hat{N}$ can be a leader, follower or equal in hierarchy, with respect to a given agent i.

3.1.2 Stackelberg Learning Objective

Each of the N agents has a continuous policy $\boldsymbol{\mu}_{\theta_i}$ parameterized by $\theta = \{\theta_1, \ldots, \theta_N\}$. We omit θ from the subscript of $\boldsymbol{\mu}_{\theta_i}$ to minimize notation resulting in the abbreviated form $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N\}$. We are considering agents with mixed cooperative and competitive objectives, hence each agent has its own objective it is trying to optimize. Given agent i, its leaders' actions, $a_l = \boldsymbol{\mu}_l(o_l), \forall l \in \mathcal{L}(i)$, are computed using private observations o_l and are unaffected by agent i's action since the leaders are already committed to their actions. The leaders of agent i may contain their own leaders whose actions are considered by $\boldsymbol{\mu}_l$, but these are considered to be part of o_l to minimize notation. The set of leader actions is $a_L = \{a_{l_1}, \ldots, a_{l_K}\}$, where $K = |\mathcal{L}(i)| < N$.

The follower responses, $a_f = \boldsymbol{\mu}_f(o_f, a_i), \forall f \in \mathcal{F}(i)$, are computed using private observations o_f and agent i's action a_i. Each follower $f \in \mathcal{F}(i)$ may have other leaders whose actions a_L, where $a_i \in a_L$ are part of the input to $\boldsymbol{\mu}_f$, but these actions are omitted to minimize notation and considered to be part of o_f. We explicitly indicate a_i as an input to $\boldsymbol{\mu}_f$ since we are formulating the objective function with respect to agent i. The set of follower actions is $a_F = \{a_{f_1}, \ldots, a_{f_M}\}$, where $M = |\mathcal{F}(i)| < N$. This formulation shows that the followers of agent i take into account a_i when selecting an action that maximizes

their objective. The resulting Stackelberg learning objective is $\max_{\theta_i} J(\theta_i)$ where

$$
\begin{aligned}
J(\theta_i) &= \mathbb{E}_{s\sim\rho^{\boldsymbol{\mu}}}[R_i] \\
&= \mathbb{E}_{\mathbf{x}^0\sim\rho}\left[\begin{array}{l} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}^0, a^0_L, a^0_i, a^0_F) \mid \\ a^0_i = \boldsymbol{\mu}_i(o^0_i), \\ a^0_L = \boldsymbol{\mu}_l(o^0_l)\ \forall l \in \mathcal{L}(i) \\ a^0_F = \boldsymbol{\mu}_f(o^0_f, a^0_i)\ \forall f \in \mathcal{F}(i) \end{array}\right]
\end{aligned} \tag{3.1}
$$

where $\rho^{\boldsymbol{\mu}}$ denotes the discounted state visitation distribution for deterministic policy $\boldsymbol{\mu}_i$, parameterized by θ_i. The follower responses a^0_F at time-step 0 are conditioned upon a^0_i. This is an actor-critic setup where the deterministic policy $\boldsymbol{\mu}_i$ is the actor and the centralized action-value function $Q^{\boldsymbol{\mu}}_{S,i}$ is the critic for each agent $i \in \hat{N}$.

The modified centralized action-value function $Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_L, a_i, a_F)$ in Eq. 3.1 can be rewritten recursively as:

$$
\begin{aligned}
Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_1, \ldots, a_N) =& r_i(\mathbf{x}, a_1, \ldots, a_N) + \\
& \gamma\mathbb{E}_{\mathbf{x}'}\left[\begin{array}{l} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}', a'_L, a'_i, a'_F) \mid \\ a'_i = \boldsymbol{\mu}_i(o'_i), \\ a'_L = \boldsymbol{\mu}_l(o'_l)\ \forall l \in \mathcal{L}(i) \\ a'_F = \boldsymbol{\mu}_f(o'_f, a'_i)\ \forall f \in \mathcal{F}(i) \end{array}\right]
\end{aligned} \tag{3.2}
$$

where the resulting Q function accounts for both the leaders and followers as it conditions on the current global state $\mathbf{x} = (o_1, \ldots, o_N)$ and actions $\{a_1, \ldots, a_N\}$. The Q-value resulting from this function represents the current reward r_i, plus the discounted future return starting from next state $\mathbf{x}'$, discounted by γ. Eq. 3.2 is equivalent to the MADDPG Q function in section 2.2.1, but with the addition of leader-follower dynamics. This definition allows us to apply off-policy temporal difference learning [25] to derive the update rule for $Q^{\boldsymbol{\mu}}_S$ as done for DQN.

3.2 Actor Critic Updates

As mentioned previously, action-value functions $Q^{\boldsymbol{\mu}}_S$ are updated using off-policy temporal difference learning as seen in section 2.1.2, while deterministic policies $\boldsymbol{\mu}$ are updated using the Deterministic Policy Gradient Theorem [31] described in section 2.1.4. In the related works, none of the actions taken by other agents depended on action a_i by agent i and

consequently θ_i. On the other hand, our Stackelberg learning objective described above contains followers whose responses depend on action a_i and θ_i. This dependence on a_i affects the gradient of the objective $\nabla_{\theta_i} J(\theta_i)$. Note that the Stackelberg objective function in Eq. 3.1 represents the expected return which we want to maximize, so we also refer to $\nabla_{\theta_i} J(\theta_i)$ as the gradient of the expected return.

3.2.1 Actor Update

In an environment with multiple hierarchies an agent i can have multiple followers with actions a_F. Each action $a_f \in a_F$ impacts the gradient $\nabla_{\theta_i} J(\theta_i)$ used to update agent i's policy $\boldsymbol{\mu}_i$. To determine the impact of each follower response, consider a two agent system with agents A_1, A_2 as leader and follower respectively. The Stackelberg objective for the leader A_1 is given by:

$$J(\theta_1) = \mathbb{E}_{\mathbf{x}\sim\rho}\left[\begin{array}{l} Q^{\boldsymbol{\mu}}_{S,1}(\mathbf{x}, a_1, a_2) \mid \\ a_1 = \boldsymbol{\mu}_1(o_1), \\ a_2 = \boldsymbol{\mu}_2(o_2, a_1) \end{array}\right] \tag{3.3}$$

The gradient $\nabla_{\theta_1} J(\theta_1)$ for A_1 with respect to θ_1, considering the impact of A_2's response:

$$\nabla_{\theta_1} J(\theta_1) = \mathbb{E}_{\mathbf{x},a\sim D}\left[\begin{array}{l} \nabla_{\theta_1}\boldsymbol{\mu}_1(o_1)\nabla_{a_1} Q^{\boldsymbol{\mu}}_{S,1}(\mathbf{x}, a_1, a_2) \\ +\nabla_{\theta_1}\boldsymbol{\mu}_1(o_1)\nabla_{a_2} Q^{\boldsymbol{\mu}}_{S,1}(\mathbf{x}, a_1, a_2)\nabla_{a_1}\boldsymbol{\mu}_2(o_2, a_1) \mid \\ a_1 = \boldsymbol{\mu}_1(o_1), \\ a_2 = \boldsymbol{\mu}_2(o_2, a_1) \end{array}\right] \tag{3.4}$$

where $\nabla_{\theta_1}\boldsymbol{\mu}_1(o_1)\nabla_{a_2} Q^{\boldsymbol{\mu}}_{S,1}(\mathbf{x}, a_1, a_2)\nabla_{a_1}\boldsymbol{\mu}_2(o_2, a_1)$ is the contribution of follower A_2 to the gradient $\nabla_{\theta_1} J(\theta_1)$, resulting from the chain rule. These three partial derivative terms make intuitive sense. The first term $\nabla_{\theta_1}\boldsymbol{\mu}_1(o_1)$ describes the updates to θ_1 needed to modify the output of $\boldsymbol{\mu}_1$ to increase the expected return. Modifying the output of $\boldsymbol{\mu}_1$ also affects the follower response since it conditions on the leader's action. This impact on the follower response due to changes in a_1 is represented by $\nabla_{a_1}\boldsymbol{\mu}_2(o_2, a_1)$. Finally, $\nabla_{a_2} Q^{\boldsymbol{\mu}}_{S,1}(\mathbf{x}, a_1, a_2)$ describes how changes to the follower response affect the resulting action-value $Q^{\boldsymbol{\mu}}_{S,1}$.

In the general case, the contribution of a follower $j \in \mathcal{F}(i)$ with action $a_j = \boldsymbol{\mu}_j(o_j, a_i)$ given $a_i = \boldsymbol{\mu}_i(o_i)$, to the gradient of the expected return of agent i is given by:

$$\frac{\partial Q^{\boldsymbol{\mu}}_{S,i}}{\partial a_j}\frac{\partial \boldsymbol{\mu}_j}{\partial a_i} = \left[\begin{array}{l} \nabla_{a_j} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_1, \ldots, a_j, \ldots, a_N)\nabla_{a_i}\boldsymbol{\mu}_j(o_j, a_i) \mid \\ a_i = \boldsymbol{\mu}_i(o_i), \\ a_j = \boldsymbol{\mu}_j(o_j, a_i) \end{array}\right] \tag{3.5}$$

Computing the partial contributions of all the follower responses a_F according to Eq. 3.5 with respect to a_i yields:

$$\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F) = \sum_{j=1}^{M} \left[\frac{\partial Q^{\boldsymbol{\mu}}_{S,i}}{\partial a_j} \frac{\partial \boldsymbol{\mu}_j}{\partial a_i} \right] \tag{3.6}$$

where $M = |\mathcal{F}(i)| \leq N$. The resulting gradient value from Eq. 3.6 allows us to generalize our method to a cooperative or competitive setting since we do not have to make any prior assumptions about the type of agents in the environment. This also removes the need to maximize over the entire action space as done by prior Stackelberg reinforcement learning models in section 2.3, as it would be computationally intractable in continuous action spaces.

Eq. 3.6 illustrates how a change to a_i impacts follower responses, which are conditioned on a_i and consequently affect the output of $Q^{\boldsymbol{\mu}}_{S,i}$. Taking the follower contributions into account, the gradient of the expected return for each agent i can be written as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x},a\sim D} \left[\begin{array}{l} \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_1, \ldots, a_i, \ldots, a_N) \\ +\nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F) \mid \\ a_i = \boldsymbol{\mu}_i(o_i), \\ a_F = \boldsymbol{\mu}_f(o_f, a_i) \ \forall f \in \mathcal{F}(i) \end{array} \right] \tag{3.7}$$

where the experience replay buffer D contains tuples $(\mathbf{x}, \mathbf{x}', a_1, ..., a_N, r_1, ..., r_N)$. Each tuple contains the current global state $\mathbf{x}$, next global state $\mathbf{x}$', all agent actions and rewards. Note that each agent contains its own objective function and it is trying to maximize its expected return. The return depends on each agent's action-value function $Q^{\boldsymbol{\mu}}_{S,i}$ which also needs to be updated.

3.2.2 Critic Update

The centralized action-value function $Q^{\boldsymbol{\mu}}_{S,i}$ is updated using off-policy temporal difference learning as defined in DQN in section 2.1.2. Though this is a centralized action-value function, each agent in our formulation maintains its own critic network as done in MADDPG in section 2.2.1. Therefore, there are a total of $2N$ neural networks being trained which include N actors and critics, represented by deterministic policies $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N\}$ and action-value functions $Q^{\boldsymbol{\mu}}_S = \{Q^{\boldsymbol{\mu}}_{S,1}, \ldots, Q^{\boldsymbol{\mu}}_{S,N}\}$ respectively. Having each agent learn a separate Q function allows agents to have arbitrary reward structures, including conflicting rewards in competitive settings [24]. In contrast, having a single centralized critic is limited

to cooperative settings as seen in section 2.2.3. Note that the centralized critic is required for the environment to remain stationary, since the policies change over time as training progresses. If the critics did not explicitly condition on the actions of all agents and the global state, then the environment would be non-stationary from the view point of any one agent. This would result in the critics being unable to make sense of the changing behaviour of other agents and consequently be unable to guide the policy training.

The Q function update is identical to the MADDPG update with the added constraint of leaders committing to their actions before followers, and allowing followers to observe and respond to this commitment. Accommodating these constraints, the Q-function is updated by minimizing the temporal difference error:

$$\begin{aligned}
\mathcal{L}(\omega_i) &= \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'\sim D}[(Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \\
y &= r_i + \gamma Q^{\boldsymbol{\mu}'}_{S,i}(\mathbf{x}', a'_L, a'_i, a'_F) \\
a'_i &= \boldsymbol{\mu}'_i(o_i), \\
a'_L &= \boldsymbol{\mu}'_l(o_l),\ \forall l \in \mathcal{L}(i) \\
a'_F &= \boldsymbol{\mu}'_f(o_f, a'_i), \forall f \in \mathcal{F}(i)
\end{aligned} \tag{3.8}$$

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}'_L, \boldsymbol{\mu}'_i, \boldsymbol{\mu}'_F\} = \{\boldsymbol{\mu}_{\theta'_1}, \ldots, \boldsymbol{\mu}_{\theta'_N}\}$ is the set of target policies with delayed parameters θ', and $Q^{\boldsymbol{\mu}'}_S = \{Q^{\boldsymbol{\mu}'}_{S,1}, \ldots, Q^{\boldsymbol{\mu}'}_{S,N}\}$ is the set of target critics with delayed parameters ω'. These target networks are updated periodically with the most recent θ and ω, which helps stabilize learning [25]. Combining Eq. 3.7 and Eq. 3.8 yields the main components of our proposed Stackelberg learning framework for hierarchical multi-agent systems, the overall algorithm is presented in Alg. 2. In its current form this algorithm struggles to learn stable policies when there are more than one followers. This challenge of learning with multiple followers is addressed in the Chapter 4, and the final updated algorithm is presented in Alg. 2.

3.2.3 Position Based Hierarchy

Our formulation thus far and the related works in section 2.3 have assumed that all agents have fixed positions within the hierarchy. This assumption does not always hold and in certain domains the hierarchy between agents can periodically change over the course of a run. One instance of this is when the hierarchy between agents is enforced by the agents' position within the environment. For example, in a racing domain the leading vehicle acts as the leader and can learn to position itself such that it prevents the tailing vehicles (followers) from overtaking its lead, while the followers are learning to overtake the leader

Algorithm 2: Stackelberg Multi-Agent Reinforcement Learning (SMARL) for N agents

for *episode* $= 1$ *to* M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state information $\mathbf{x}$
  for $t = 1$ *to max-episode-length* **do**
    for $h = 1$ *to max-hierarchy-depth* **do**
      for *each agent* $i \in \mathcal{H}(h)$ *in the current level* h **do**
        select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i, a_L) + \mathcal{N}_t$ w.r.t. the current policy, exploration and a set of leader actions a_L
      end
    end
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$, and set $\mathbf{x} \leftarrow \mathbf{x}'$
    for *agent* $i = 1$ *to* N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^k, a^k, r^k, \mathbf{x}'^k)$ from $\mathcal{D}$
      Set $y^k = r_i^k + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^k, a'_1, \ldots, a'_N)\big|_{a'_j = \boldsymbol{\mu}'_j(o_j^k), a'_f = \boldsymbol{\mu}'_f(o_f^k, a'_i), \forall f \in F(i)}$
      Update critic by minimizing the loss

$$\mathcal{L}(\theta_i) = \frac{1}{S} \sum_k \left(Q_i^{\boldsymbol{\mu}}(\mathbf{x}^k, a_1^k, \ldots, a_N^k) - y^k \right)^2$$

      Update actor using the sampled policy gradient with $\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_F)$ defined in Eq. 3.6

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_k \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^k) \big(\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^k, a_1, \ldots, a_N) + \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_F) \big)\big|_{a_i = \boldsymbol{\mu}_i(o_i^k), a_f = \boldsymbol{\mu}_f(o_f^k, a_i), \forall f \in F(i)}$$

    end
    Update target network parameters for each agent $i : \theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
  end
end

[22]. This creates a problem where agents collect a disproportionate amount of samples for one position over another within the hierarchy, resulting in the algorithm being sample inefficient. This inefficiency is with respect to the experience replay buffer D, described in section 3.2.1. If an agent collects the majority of its experience from a hierarchy level $h \in \hat{H}$, then it may only perform well on level h. Agents may even fail to learn any decent policies and require many more samples to converge to a good policy.

We address this issue by mapping policies to positions within the hierarchy instead of individual agents in the environment. More concretely, we set $|\boldsymbol{\mu}| = H \leq N$, where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_H\}$ is a set of deterministic policies for each level in the hierarchy set $\hat{H}$, and the experience replay buffer stores samples based on hierarchy instead of individual agents. This allows for efficient use of samples during training and learning specialized policies for each position in the position based hierarchy. In addition, this method scales well when the total number of agents greatly exceeds the total levels of the hierarchy $N \gg H$ since the set of policies stays fixed $|\boldsymbol{\mu}| = H$.

3.3 Summary

In this chapter we present the Stackelberg learning objective for continuous action spaces with multiple levels of hierarchy, along with the method to optimize this objective. We first detail the different inputs to the policies based on positions within the hierarchy in section 3.1.1. Hierarchies in our model relate to the ability to commit to an action before others; agents higher in this hierarchy can commit their actions before agents lower within the hierarchy. We also discuss position based hierarchies, their impact on policy training and methods to deal with agents periodically transitioning between the different hierarchy levels in section 3.2.3. The main contributions in this chapter are the formulation of the Stackelberg learning objective in sectoin 3.1.2, followed by the actor-critic updates in sections 3.2. Our Stackelberg learning objective demonstrates a method for handling multiple leaders and followers with continuous action spaces. We also detail our method for optimizing the Stackelberg learning objective with asymmetric information using policy gradients and temporal difference learning.

Chapter 4

Follower Influence on Gradient Updates

The Stackelberg Multi-Agent Reinforcement Learning (SMARL) algorithm we formulated in Chapter 3 struggles to learn stable policies as the number of followers increase. In this chapter, we look at the cause of this issue and propose methods to resolve this problem, while learning policies which are consistent with the Stackelberg model. In addition to addressing the main issue, we also investigate the effect of follower influence on leader policy convergence.

4.1 Multiple Followers Problem

In the previous chapter we demonstrated the relation between leaders and followers, where leaders commit to their actions before the followers, but followers get to observe this commitment. We illustrate this process in section 3.1.1, where followers' policies take as input the leader action $\boldsymbol{\mu}_f(o_f, a_i)$ for some leader i. Using this setup, we formulate the Stackelberg learning objective and use actor-critic methods to train policies which optimize their objectives. The actor update requires computing the gradient of the objective function with respect to each agent's policy parameters, $\nabla_{\theta_i} J(\theta_i)$. Now since our learning objective had follower policies condition on leader actions, the gradients of each leader's objective is dependent on those follower policies. Specifically, as we detailed in section 3.2.1, the contribution of a follower $j \in \mathcal{F}(i)$ with action $a_j = \boldsymbol{\mu}_j(o_j, a_i)$ given $a_i = \boldsymbol{\mu}_i(o_i)$, to the

gradient of the expected return of agent i is given by:

$$\frac{\partial Q^{\boldsymbol{\mu}}_{S,i}}{\partial a_j}\frac{\partial \boldsymbol{\mu}_j}{\partial a_i} = \left[\begin{array}{l} \nabla_{a_j} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_1, \ldots, a_j, \ldots, a_N)\nabla_{a_i}\boldsymbol{\mu}_j(o_j, a_i) \mid \\ a_i = \boldsymbol{\mu}_i(o_i), \\ a_j = \boldsymbol{\mu}_j(o_j, a_i) \end{array}\right]$$

The partial contributions of all the follower responses a_F with respect to a_i yields:

$$\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F) = \sum_{j=1}^{M}\left[\frac{\partial Q^{\boldsymbol{\mu}}_{S,i}}{\partial a_j}\frac{\partial \boldsymbol{\mu}_j}{\partial a_i}\right] \tag{4.1}$$

This above term describes the change in output of $Q^{\boldsymbol{\mu}}_{S,i}$ due to changes in follower response with respect to a_i. Since $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F)$ holds the cumulative updates corresponding to each follower $f \in \mathcal{F}(i)$, as the number of followers increase, this term starts to dominate the primary update term $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_i, a_{-i})$, where a_{-i} is the set of all actions not including a_i. The primary update describes the change in $Q^{\boldsymbol{\mu}}_{S,i}$ with respect to a_i and is equivalent to the MADDPG actor update seen in section 2.7. The average of the actor loss throughout training, represented by $\mathcal{L}_f$, $\mathcal{L}_p$ to compute gradients $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F)$ and $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_i, a_{-i})$ respectively are shown in Table 4.1 below for different number of total agents:

	avg. $\mathcal{L}_p$	avg. $\mathcal{L}_f$	% diff.
$N = 3$	109.61 $\pm$ 55.4	123.12 $\pm$ 59.96	11.60
$N = 4$	132.04 $\pm$ 70.04	148.15 $\pm$ 71.18	11.47
$N = 5$	146.31 $\pm$ 141.71	169.52 $\pm$ 147.24	14.70

Table 4.1: Primary avg. loss $\mathcal{L}_p$ and follower response avg. loss $\mathcal{L}_f$ for Particle Tag environment from the leader's perspective using fixed size actor-critic networks.

This table shows the average actor loss with respect to a single leader throughout training. Primary average loss is equivalent to the MADDPG actor loss, while the follower response average loss results from our SMARL objective 3.1. If the followers did not condition on the leader's action a_i, the two values would be equal since the critics would produce the same output. This is not the case, since followers do react to the leader's enforcement, resulting in varying critic outputs. Using a fixed size actor and critic network, the loss values and their standard deviation increase as the total number of agents increase. This is mainly due to the increase in state space, but the follower response loss has a larger overall increase compared to the primary loss due to the increase in total number of followers. This has a direct impact on the gradient computation as shown in Table 4.2.

	$\|\nabla Q_p\|_1$	$\|\nabla Q_f\|_1$	% diff.
$N = 3$	1.5784E-02	7.7025E-03	68.82
$N = 4$	2.099E-02	7.8176E-03	91.45
$N = 5$	3.5345E-02	9.1062E-03	118.06

Table 4.2: L1 norm of primary and follower response gradients computed using $\mathcal{L}_p$ and $\mathcal{L}_f$.

These values in Table 4.2 are used to illustrate the overall magnitude of the gradient updates throughout training. They are computed by taking the mean of the gradient, used to update the actor network, at each time-step and computing the L1 norm across all training steps. The resulting values show that increasing total number of agents leads to, on average, larger changes in the weights of the actor network. This is expected since the average loss also increased with total number of agents as shown in Table 4.1. Note that the norm of the follower response gradient update is also fairly significant and will dominate the primary update as the number of followers increase. In an effort to soften the influence of the follower response we propose scaling its gradient.

4.2 Scaling Follower Gradient Contribution

In order to soften the follower gradient contributions, we propose scaling the follower gradient component $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F)$ by α/M, where $M = |\mathcal{F}(i)|$, and $\alpha \in [0, 1]$ is a tunable coefficient. We first normalize the gradient by dividing by the max number of followers for agent i. This is important because each leader could have different number of followers and a fixed α value would not achieve consistent effect across different leader updates. This hyper-parameter α gives us more control over how much of the follower response we want to consider during actor policy updates. The updated but shortened version of the actor update from Eq. 3.2.1 is presented below:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x}, a \sim D} \begin{bmatrix} \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \Big[\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_i, a_{-i}) + \frac{\alpha}{M} \nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F) \Big] \\ \vdots \end{bmatrix} \tag{4.2}$$

Note that when $\alpha = 0$ the gradient update $\nabla_{\theta_i} J(\theta_i)$ is equivalent to the MADDPG actor update, but it is not exactly equal to the MADDPG update since our agent policies are modified to incorporate the Stackelberg model. Using this scaling factor attenuates the effect of follower policies on the leader's policy gradient updates, but still leads to unstable learning behaviour due to the changing scale of the gradients.

4.2.1 Normalizing Follower Gradient Contribution

When policies are being trained, the scale of the gradient changes based on the expected return $J(\theta_i)$. Updates to θ_i could result from a small change in $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_i, a_{-i})$ with respect to a_i, but a relatively large change in $\nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F)$. This is evident in the leader and follower losses presented in Table 4.1. In such scenarios, the follower contributions to the gradient can still overwhelm the primary update, even with a scaling factor α/M. This problem of varying gradient scales was encountered in M3DDPG [20, 9] described in section 2.2.2, where their suggested workaround was to normalize the gradients. Using this suggestion, we propose normalizing the follower gradient contributions as follows:

$$
\begin{aligned}
g &= \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F) \\
\hat{g} &= \frac{\alpha}{M} \frac{g}{\|g\|}
\end{aligned}
$$

where g is the original follower gradient from Eq. 3.7 and $\hat{g}$ is scaled and normalized version. We use the L2 norm $\|g\|$ to normalize g in order to prevent sparsity in $\hat{g}$, since we simply want to soften the impact of g and not filter any of its parameter updates. Using $\hat{g}$ the final updated version of the actor update from Eq. 3.7 is as follows:

$$
\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x}, a \sim D} \left[\begin{array}{l} \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_i, a_{-i}) + \frac{\alpha}{M} \frac{g}{\|g\|} \mid \\ g = \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{S,i}(\mathbf{x}, a_F), \\ a_i = \boldsymbol{\mu}_i(o_i), \\ a_F = \boldsymbol{\mu}_f(o_f, a_i) \ \forall f \in \mathcal{F}(i) \end{array} \right] \tag{4.3}
$$

Normalizing these follower gradient contributions allows us to attenuate their impact on the overall gradient update of the actor networks. This gives us more freedom to experiment with the scaling factor α introduced in the previous section. This value does not have to remain fixed during training and can be updated to control the amount of influence the follower contributions have on the actor network updates.

4.3 Investigating Follower Influence

In our proposed SMARL algorithm and related works discussed in section 2.2, all agent policies are trained from scratch. This means that all agents start with policies which contain randomly initialized parameters θ_i. This allows them to explore the environment and collect samples in a wide range of states which might lead to better decisions in the future

that benefit policy training. On the other hand, when followers are performing such exploration, their actions are not always in response to the leader's enforcement. In such cases, follower responses act as noise to the leader's policy update and slow convergence. The follower responses only become beneficial to the leader policy updates when the followers start reacting rationally to the leader enforcement as required by the Stackelberg model. This occurs later during training when the agents start exploiting the action-value function by selecting the best action suggested at each timestep and avoiding random actions for exploration.

Using this understanding of how the policy training evolves over time, we propose adjusting the gradient scaling coefficient α throughout the training. Specifically, we propose five different schedules for the α coefficient which start low and increase as policy training progresses. These schedules include a linear, step, exponential, logarithmic and sigmoid functions; they are illustrated in Fig. 4.1 below. Most of the schedules are designed to minimize the effect of follower gradient contribution to the leader policy update early during training, and increase the follower contributions at different rates as the training progresses. This allows leader policies to be updated while minimizing the noise from follower responses. The logarithmic update is different from other schedules as it has a faster increase early during training before slowly approaching the max α value. The exponential and linear schedules are fixed at 1.0 near the end of training since they do not have an upper limit. These schedules enable actor networks to be trained using information from multiple follower responses.

4.4 Summary

This chapter addressed the problem of learning stable policies while accounting for multiple followers. We first discuss how a leader's policy update is affected by the number of followers due to our Stackelberg learning objective in section 4.1. The gradient contribution of the follower responses starts to dominate the primary update signal as the number of followers increases. To address this problem we propose scaling the follower gradient contribution in section 4.2. This is done by first normalizing the follower contribution by the total number of followers for a given leader and then scaling by a tunable coefficient α. Since the scale of the gradient changes throughout training based on the expected return, we also normalize the follower gradient using the L2 norm. Lastly in section 4.3, we investigate how varying α throughout the training process could lead to better convergence and propose different α update schedules for testing. The next chapter presents environments to test our proposed SMARL algorithm against the baseline MADDPG algorithm along with the results.

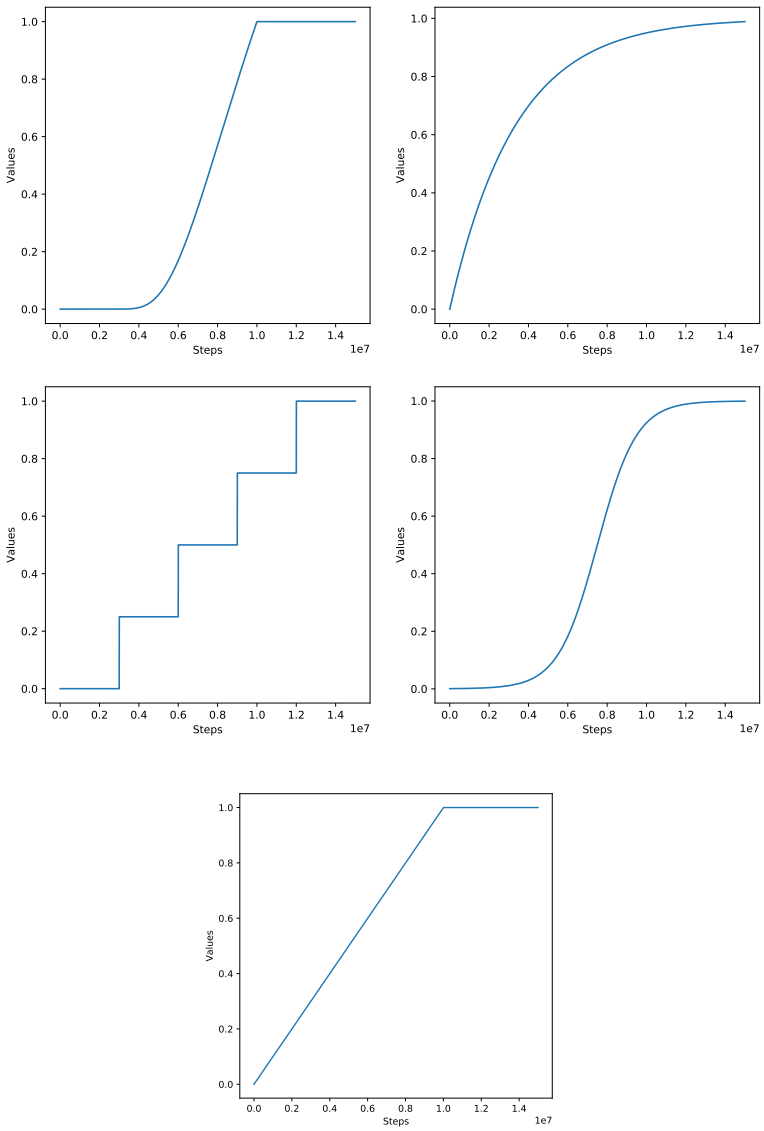

Figure 4.1: Variable α update schedules for controlling magnitude of follower influence on gradient updates.

Chapter 5

Experiments

This chapter presents some multi-agent experimental domains in which we test our proposed Stackelberg Multi-Agent Reinforcement Learning (SMARL) algorithm. We compare our proposed algorithm to MADDPG [24] which acts as the baseline. The environments used include a subset of the multi-agent particle environments [26, 24] and a custom multi-agent highway driving environment. The particle environments were originally used in [24] to test their MADDPG algorithm. We only use a subset of these, which have clear leader-follower dynamics. The following sections discuss details of the test domains, training configuration and performance results.

5.1 Environments

The proposed SMARL algorithm is only applicable in multi-agent domains due to the use of the Stackelberg model. Specifically, the use of this algorithm is only relevant to domains where the interaction between agents can be modeled by leader-follower dynamics. Without the presence of such a hierarchy all agents would be treated as leaders and the algorithm would be equivalent to MADDPG. To test our proposed method in a hierarchical setting we built a new multi-agent highway driving simulator as it has clear definitions for leaders and followers based on position and also allows us to test general sum coordination scenarios. In addition, we also tested using the Predator-Prey and Keep-Away scenarios from the particle environments by assigning certain agents to be leaders and others to be followers. These three environments are detailed in the following subsections.

5.1.1 Multi-Agent Particle Environments

Particle environments contain N agents and L landmarks in a two-dimensional world with continuous observation and action spaces. The time between states is discretized with time-steps of 0.1 seconds. We focus on two environment called Keep-Away and Predator-Prey and provide details of each below.

Predator-Prey

In our experiments this scenario consists of three cooperative predator agents whose goal is to chase a faster adversary (prey) around a randomly generated environment with two landmarks impeding the way. This setup is illustrated in Fig. 5.1. The predators receive a negative reward proportional to the sum of the total Euclidean distance of all predators to the prey and a larger positive reward for every predator colliding with the prey. Since each predator's reward depends on the actions of other predators, their rewards are shared, which promotes cooperative behaviour. On the other hand, the prey is given a small reward proportional to the sum of total Euclidean distance from each predator, but a large negative reward for each collision with a predator. The prey is also penalized for leaving the target area to prevent it from exploiting soft boundaries in the particle environment. These soft boundaries exist to prevent the predators from cornering the prey and promote movement.

The original scenario setup does not contain any hierarchies since the private observations for each agent are equivalent and rewards enforce a zero-sum game. We add asymmetry into the scenario by allowing the prey to tune its action based on anticipated reactions from the predators. This is done by modifying the objective function as detailed in Section 3.1. We designate the prey as the leader by allowing it to commit to its action before the predators. Predators are appointed as followers who can observe the prey's action and react accordingly. This results in a two level hierarchy with the prey on one level and all the predators on a level below the prey. Each agent's private observation contains velocity, position, relative landmark positions, relative positions and velocities of other agents; predators also observe the prey's action in addition to the default observation.

Keep-Away

In our configuration of this scenario we use two landmarks to represent a target and placeholder, one agent whose goal is to reach the target landmark and two cooperative

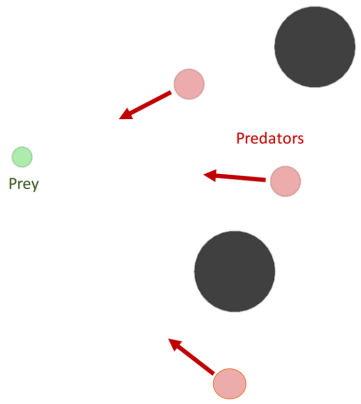

Figure 5.1: Predator-Prey environment containing two obstacles, three predators (followers) and a prey (leader).

adversary agents whose goal is to prevent the lone agent from reaching the target landmark while also remaining close to the target. This is illustrated in Fig. 5.2. We setup a two level hierarchy by designating the lone agent as the leader, allowing it to commit to its action before the adversaries. Consequently, the adversaries are labeled as followers and can observe the leader's action. The leader is rewarded the negative of its Euclidean distance to the goal, hence it tries to reduce this distance. The followers' reward is composed of the leader's distance to the target minus their own distance to the target landmark; it is not shared as is done in Predator-Prey.

The original scenario used by [24] already contains asymmetric information between agents prior to our labels of leader and followers. This is because the leader knows its distance to the target while the followers need to infer it from the movement of that agent. The leader's reward also provides information about the target destination, while followers obtain mixed feedback. However, we want to test how asymmetry in the environment can be exploited by agents at the top of the hierarchy to influence the actions of agents at the bottom; this requires controlling the source of asymmetry in the scenario. We do this by making the leader and followers receive similar information in their private observations, with the addition of leader's action to the followers' observation. All agents only see their current physical states, and relative positions of other agents and landmarks, but followers also observe the leader's action.

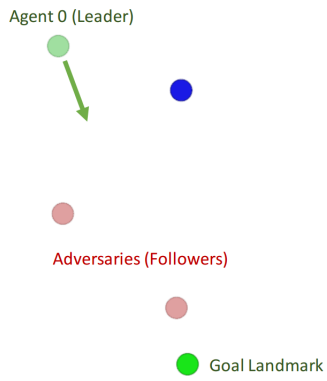

Figure 5.2: Keep-Away environment with two adversaries, two landmarks and a leader.

5.1.2 Highway Driving Environment

In order to test general sum games and positions based hierarchies, we propose our new multi-agent highway driving environment. This environment contains a straight three lane driving area, three obstacles and two agents. The obstacles are vehicles that are not being trained, but are initialized with a random position and velocity. The agents are also vehicles, but are controlled by their respective policies. The main goal for each agent is to travel as fast as possible while avoiding crashing into others. This general setup is illustrated in Fig. 5.3. There are two levels of difficulty in this environment labeled as Driving Easy and Driving Hard. The easier scenario always keeps the obstacles in the centre of the lane, while the harder scenarios have obstacles initialized with random positions within the lane. This small change makes training more difficult since agents cannot learn a policy which simply travels between the lanes. Agents control their actions using two continuous values ranging from $[-1, 1]$, which represent acceleration and steering. Each run is 60 simulated seconds long and the agents can perform four actions per second (4Hz); this gives 240 steps per run. We use a clock rate of 36Hz which gives a discretized time-step of 27 milliseconds. This specific time-step size is used to allow for fast and smooth simulation. If we were to use a step-size of 0.1 seconds as in the particle environments, the simulated vehicles would not have smooth transitions.

In this environment all agents are trying to maximize their rewards and do not benefit from hindering the other agent's progress. The reward function is simply the normalized velocity of the agent offset by 1 and ranges from $[-1, 0]$. A reward of -1 corresponds to vehicle not moving, and 0 when it is moving at max velocity. A negative reward is used to motivate the agents to move at their max velocity. Agents need to learn to coordinate their

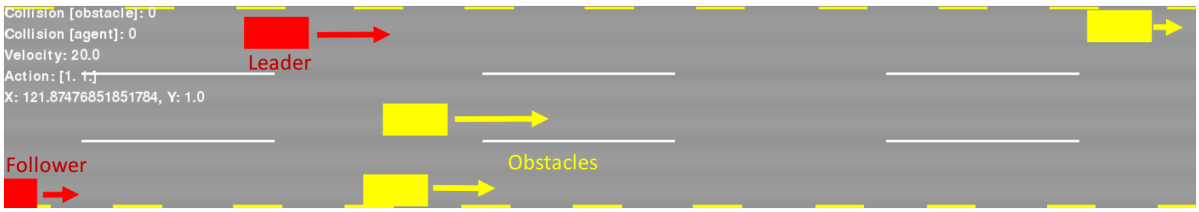

Figure 5.3: Highway driving environment with three obstacles and two agents.

actions in certain situations so that they do not crash into one another. This results in a position based hierarchy where agents physically in the lead act as the leader and agents tailing are the followers. We limit the scope of our experiments to two agents as it allows us to test the convergence between leader and follower without the overhead of training numerous agents. Leading agents have more freedom to take aggressive actions since they know followers are rational and would modify their action to maintain their positive reward. Each agent's private observation contains acceleration, steering, position, velocity, other agents' relative positions and velocities, the follower also observes the leader's action. We present the setup and results of our experiments in the next section.

5.2 Results

The actor-critic networks in our experiments use a standard multi-layered perceptron (MLP) with fully-connected layers; the network architecture for the particle and highway environments are summarized in Table 5.1. We use ReLU activation, batch norm with a batch size of 1440, discount factor of 0.99, actor learning rate of 0.0001 and critic learning rate of 0.001. Adaptive noise is added to the network weights during training to encourage exploration, but not used during testing. Particle environments are rolled out for 75 steps, while highway driving simulations run for 240 steps to simulate one minute of driving. Particle environment networks are the same size as those used in MADDPG. We use larger networks for driving environments since they contain more dynamic elements with the added constraint of coordination without collision, and larger networks were found to have better returns during our experiments.

We measured the number of training steps performed per second for each environment using the baseline (MADDPG), SMARL with static α and SMARL with variable α models; these results are presented in Table 5.2. This table shows that there was no significant change in training time between the models for particle environments. Our model resulted

	Highway Env	Predator-Prey	Keep-Away
Input Layer	$6N + 2$	$4N + 6$	$2N + 8$
Hidden Layers	4	2	2
Num. Neurons/Layer	256	128	128
Activation	ReLU	ReLU	ReLU
Output Layer	2	2	2

Table 5.1: Actor-Critic network architecture summary for all environments with N agents.

in longer training time for the easy driving scenario, but faster training time for the hard driving scenario. The main reason SMARL is slower to train than the baseline is because the agents need to be sorted by position at each step since the hierarchy is position based. On the other hand, SMARL is faster than the baseline for the hard driving scenario. This occurs because the follower agent struggles to learn a stable policy early during training using the baseline model; this causes the follower to lag behind the leader resulting in multiple obstacles being present in the simulation and hence slowing down computation. The follower agent in the easy scenario does not suffer from this issue and is able to keep a closer distance to the leader; hence, reducing the number of obstacles present which result in faster computation. The cause of these differences is visible in the learning plots presented in the next section. We also demonstrate the inverse relation between the total number of

	Baseline	Static α	Variable α
Particle Tag	73.90 ± 0.88	74.74 ± 1.00	71.30 ± 0.79
Particle Push	104.39 ± 1.70	106.43 ± 1.44	100.90 ± 1.41
Driving Easy	121.43 ± 4.23	105.14 ± 3.04	101.28 ± 2.75
Driving Hard	74.74 ± 3.28	103.12 ± 3.55	101.35 ± 3.33

Table 5.2: Average training steps/second

agents N and the training time in Figure 5.3. We use the Particle Tag environment as it always has a fixed number of entities in the environment during the simulation. The next section details the results obtained during the experiments.

	$N = 3$	$N = 4$	$N = 5$
Particle Tag (Static α)	103.64 ± 0.94	74.74 ± 1.00	50.23 ± 1.27

Table 5.3: Particle Tag average training steps/second for total agents $N = [3, 5]$

5.2.1 Highway Driving Environment Results

This section presents the reward plots for all the leader and follower agents trained to maximize reward in the driving environments. Each figure displays the means and standard deviation of the reward curves over three runs. The reward curves are used as the performance metric because the policies are trained to maximize their expected reward described by the objective function detailed in section 3.1.2. Actor policies that can achieve higher average reward over the course of a simulation run are thus better than policies which receive a lower average reward. There are three different figures for each experiment which consist of SMARL with static alpha values, SMARL with variable alpha values and a comparison between the baseline and two SMARL experiments.

Driving Easy

The performance of SMARL using static alpha values ranging from $\alpha = [0.0, 1.0]$ with increments of 0.25 for the easy driving scenario are shown in figure 5.4. The reward curves for SMARL using variable alpha schedules are shown in figure 5.5 and figure 5.6 shows the best reward curves comparing the MADDPG baseline with SMARL. In figure 5.4 we can see that the leader's reward curves are practically identical and converge to a reward around -50. This tells us that regardless of the follower gradient scaling factor α, the leaders learned a policy which on average obtained a reward of -50. The follower policies are not directly affected by α, but are included for the completeness of the results. There is more variance in the rewards obtained by the followers, but they are all on average less than those achieved by the leaders.

The reward curves for the leader using variable α schedules in figure 5.5 were similar to the results achieved by leaders in figure 5.4 using the static α values with respect to the converged reward value. A noticeable difference between the schedules is that the sigmoid and step update curves resulted in higher variance before converging to a stable leader reward value. As for the follower curves in figure 5.5, the rewards were steadily increasing but did not have time to converge in the allocated episodes; the follower reward curve for step schedule diverged near the end of training. Though the follower policies are not directly dependant on α, they are forced to react to the leader's action which is a function of α. A sudden increase in α due to the step schedule can result in a sudden change in the leader's action which the follower policy was not trained to handle, leading to diverging behaviour. Such divergence is more prevalent in later training episodes, when policies are moving from exploring to exploiting the environment dynamics, and a sudden change in the leader's action is equivalent to adding noise to the follower policy observations.

The follower policies trained using SMARL with variable α schedules performed better than those with static α values. We can see in figure 5.5 that most of the curves are comparable to each other and close to the same value as opposed to the larger spread in figure 5.4. Follower policies trained under the sigmoid and linear schedules also achieved average rewards comparable to the reward achieved by the leaders.

A comparison between the baseline MADDPG, SMARL with static and variable α is presented in figure 5.6 for the easy driving scenario. We plotted the three best runs out of five for each algorithm, and can see that SMARL policies achieved better average rewards for leader curves than the baseline. SMARL with linear α update schedule performed best overall with a significant advantage over the baseline and a smaller gain over static $\alpha = 1.0$. In terms of follower policies, SMARL[Linear] also achieved better performance than the baseline, but SMARL[$\alpha = 1.0$] did not. This result in figure 5.6 supports our reasoning for including schedules that varied α over the course of training as described in section 4.3, since this method was able to train better policies in this instance.

Driving Hard

The reward curves for SMARL with static and variable α along with a comparison to the baseline, for the hard driving scenario, are presented in figures 5.7, 5.8 and 5.9 respectively. Looking at figure 5.7, it is clear that this difficult environment resulted in lower overall average rewards across all runs compared to those in figure 5.4 for the easier scenario. There was also more variance and policy divergence for both leader and follower runs. The majority of the follower runs diverged and were unable to show any signs of convergence.

In comparison, policies trained using variable α were more stable with respect to the leaders, as shown in figure 5.8, where only the step schedule diverged. The use of variable α allowed the leader policies to show signs of stability and convergence earlier than those in figure 5.7. Follower policies on the other hand were more sensitive to the changes introduced by these schedules. The step schedule performed poorly, which was consistent with the results in figure 5.5 for the easier driving scenario. Exponential and sigmoid schedules also performed poorly closer to the end of training. The commonality between these three schedules is their sudden increase in α.

The easy driving scenario was not affected by the sudden but smooth rise in α caused by exp and sig, but the hard driving scenario was more sensitive to these sudden changes and resulted in divergence during training. Note that in figure 5.8, the follower step and exp runs both start to diverge around a point during the training which corresponds to the schedules being capped at $\alpha = 1.0$ in figure 4.1. This act of capping α does not negatively

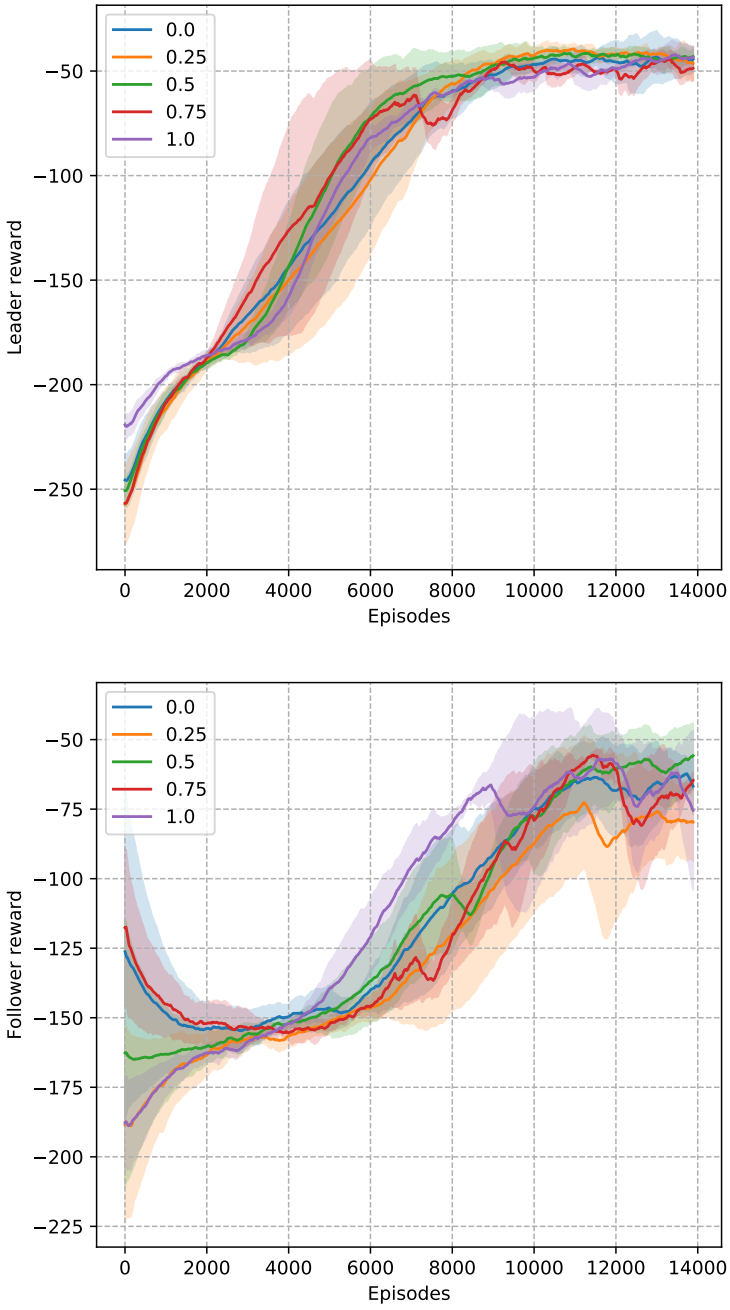

Figure 5.4: Driving Easy average rewards using SMARL with static α values

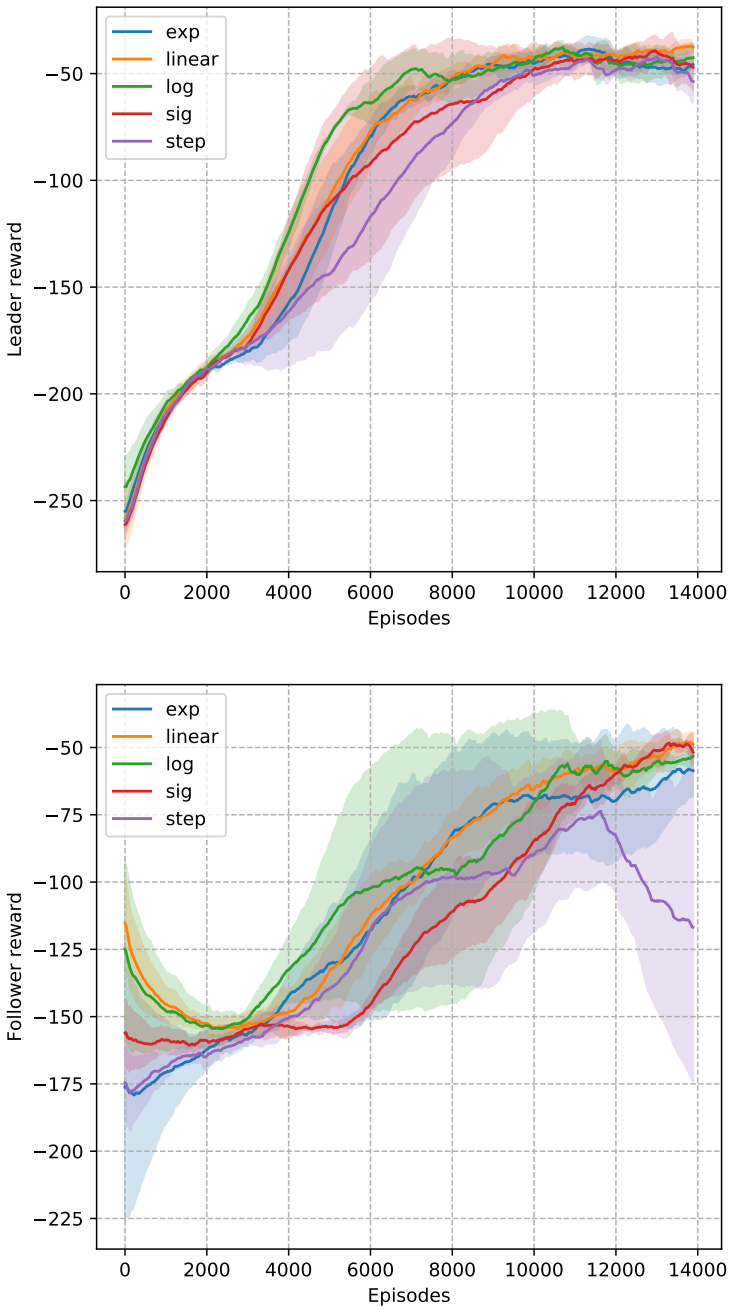

Figure 5.5: Driving Easy average rewards using SMARL with variable α values

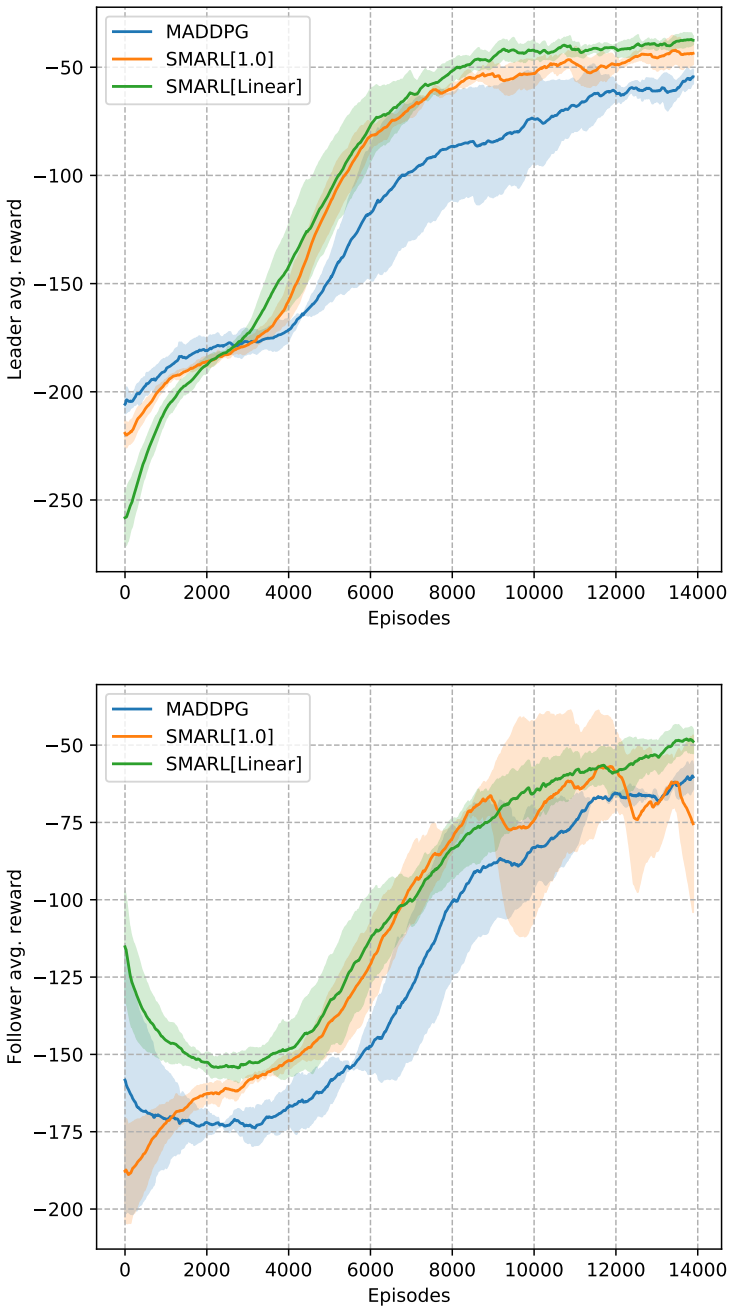

Figure 5.6: Driving Easy Baseline vs. SMARL static α vs. SMARL variable α

impact the linear schedule because a steady increase of α introduced by this linear schedule allows policies to adapt to changing signals over the course of training. A sudden increase in α caused by exp, sig and step leave the policies in a fragile state sensitive to changes and prone to divergence. Follower policies also take longer to converge than leader policies since they are learning to react to the leader's enforcement, which is changing over time.

The best run from figure 5.7 and 5.8 is compared against the best baseline (MADDPG) run and presented in figure 5.9. The plots clearly show that SMARL was able to train better leader policies that achieved higher average rewards than the baseline. SMARL with linear α update schedule was able to achieve a higher reward faster than the policies trained using static α, but they were comparable near the end of the training. One of the downsides of SMARL is that the algorithm takes longer to converge than the baseline; this is visible in the leader reward curves of figure 5.9 where the baseline has a fast rise and is steady while SMARL continues to increase. This is expected since SMARL makes use of more information made accessible through the leader follower dynamics. Analysing the follower policies we see a similar trend were the baseline converges to some value and SMARL slowly catches up to this threshold. SMARL follower policies manage to surpass the baseline average reward, but very late in the training and with high variance.

Based on these highway driving environments, we demonstrate that SMARL can outperform the baseline algorithm but require longer training time to converge to some local optimal policy. Specifically, SMARL with a linear α update schedule obtained the best performance in both the easy and hard driving scenarios, while SMARL with static $\alpha = 1.0$ was a close second in most cases. These environments tested SMARL's ability to train good policies in position based hierarchies, described in section 3.2.3, particle environments test the algorithm in fixed hierarchies.

5.2.2 Particle Environments

In this section, we present the reward plots for all leader and follower agents trained to maximize rewards for the particle environments. Similar to section 5.2.1, we present the mean and standard deviation of rewards computed over three training runs. Three plots are included for each environment illustrating the performance of SMARL with static and variable α, along with a comparison of these two methods to a baselines model.

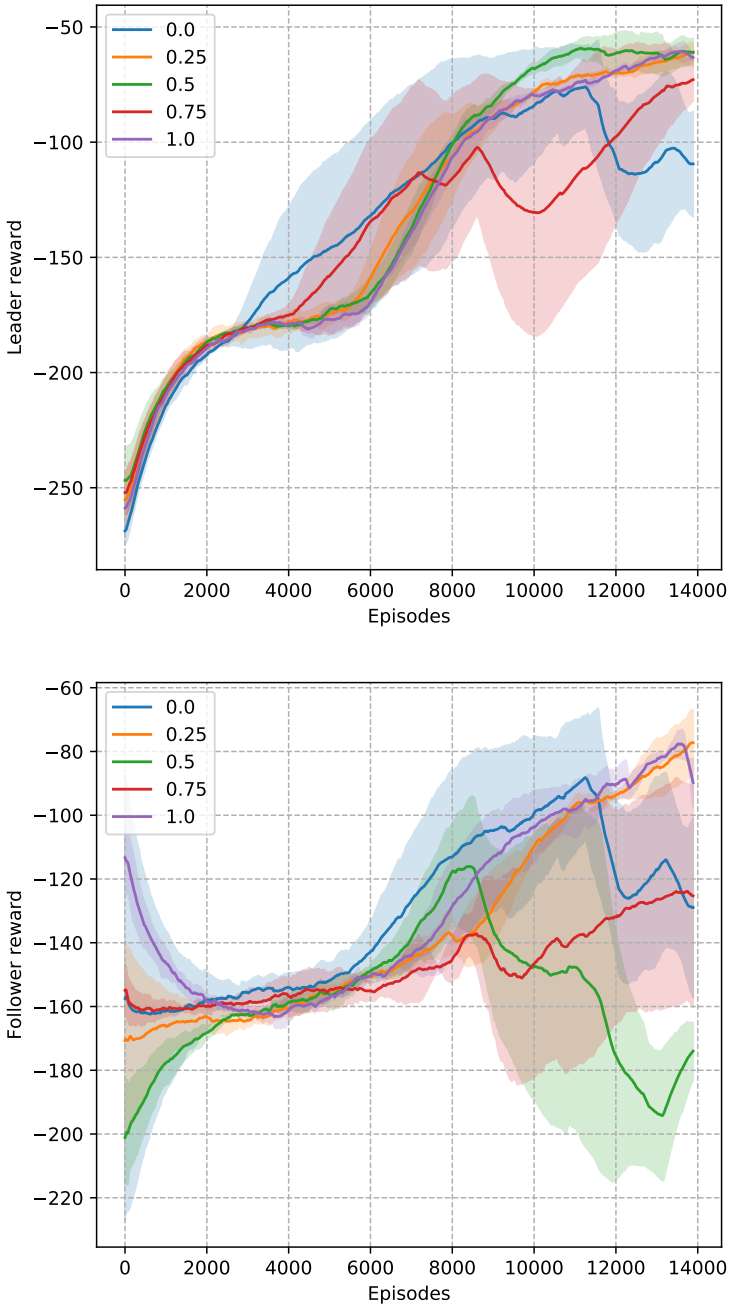

Figure 5.7: Driving Hard average rewards using SMARL with static α values

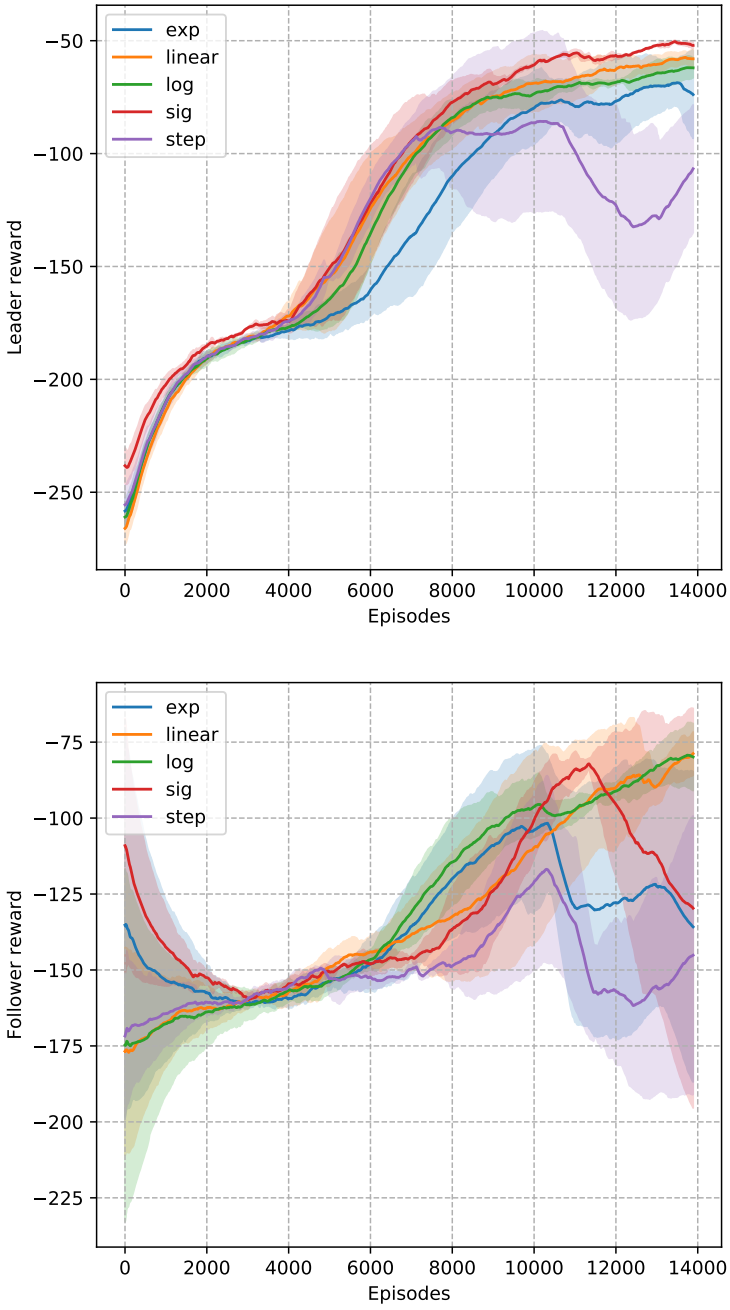

Figure 5.8: Driving Hard average rewards using SMARL with variable α values

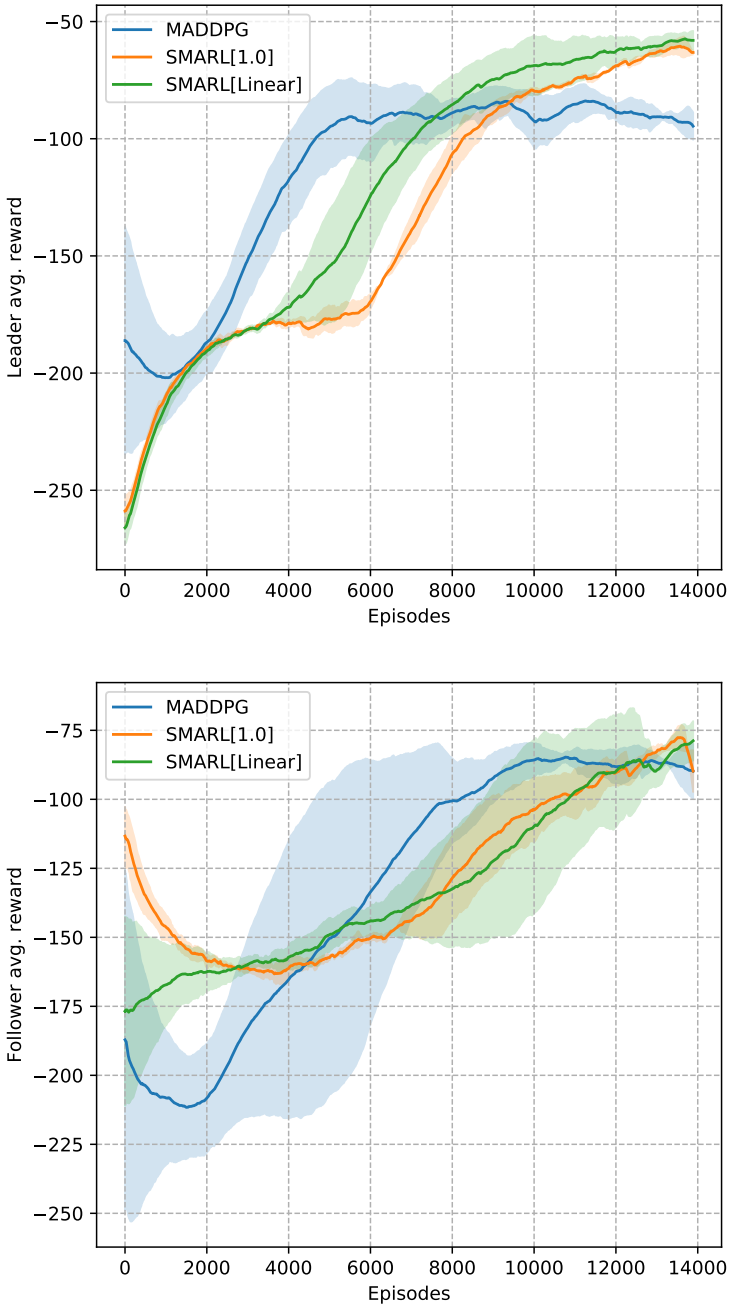

Figure 5.9: Driving Hard Baseline vs. SMARL static α vs. SMARL variable α

Predator-Prey

This section discusses the results obtained by leader and follower policies trained using SMARL in the predator-prey particle environment [26, 24]. Specifically, we look at results obtained using SMARL with static and variable α values, along with a comparison to the baseline in figures 5.10, 5.11 and 5.12 respectively. The leader and follower reward curves in figure 5.10 are not cleanly separated as they were in the driving environment and contain more overlap. The leader reaches its convergence value very early during the training since it is simply learning to run away from the predators. While the follower policies, who share a global reward, learn to coordinate their actions and settle around a similar reward value for all variations of α. Different values of α experience large variance at different points during the training, but lower α values show less overall variance than those with higher α. Specifically, $\alpha = [0.5, 0.75, 1.0]$ contain large swings during training for both leader and follower policies, with $\alpha = 0.75$ diverging near end of training.

The rewards for predator-prey policies trained using SMARL with variable α are presented in figure 5.11. These reward curves have much more separation than those obtained with static α for both leader and follower. We can see that the log and exponential update schedules struggled with high variance and were unable to converge. This can be attributed to the fast rise in α values, similar to the effect seen in the driving environments. The negative relation between the fast rise in α and instability in policy behaviour can also be seen in the sigmoid and step reward curves. This relationship is very noticeable with the step schedule, since its reward curves contain pockets of high variance corresponding to a sudden increase in α. In comparison, the variation in rewards for linear schedule is more stable throughout training.

The policies trained using SMARL with $\alpha = 1.0$ and a sigmoid schedule are compared to the MADDPG baseline method in figure 5.12. The figures show that SMARL outperforms the baseline for both leader and follower policies by a large margin. Specifically, we can see that the baseline leader policy gets worse reward as training progresses while the SMARL leader policies stay consistently higher. This is because the SMARL leaders have knowledge about follower reaction functions through our SMARL objective function described in section 3.1. Additionally, followers observe the actions taken by leaders and react accordingly to maximize their return; hence, follower policies are able to obtain higher rewards than policies trained using MADDPG. The performance of both SMARL methods is comparable as they achieve similar average rewards. However, SMARL with $\alpha = 1.0$ performed well for both leader and follower policies, while the variable sigmoid schedule obtained better leader policies than follower policies.

These results for the predator-prey particle environment show that SMARL can outper-

form the MADDPG baseline in its own test domain. These results were obtained using the default parameters for both sets of algorithms without additional hyper-parameter tuning apart from the follower gradient scaling factor α. Our results show that exploiting the hierarchies within the environment by making use of follower reactions and leader enforcement can help train better policies.

Keep-Away

This section presents the results obtained in the keep-away particle environment [26, 24] described in section 5.1.1. We present the average reward curves obtained using variable α update schedules in figure 5.13, along with a comparison between SMARL and baseline in figure 5.14. We do not present the static α results for this environment, as done for environments in previous sections, since most did not perform well; instead, we present the best static α run in the comparisons figure. In this environment the rewards were not shared between followers, therefore, each figure contains the reward curves for a leader and two followers. These reward curves start at episode 5000 because prior to this point during training the rewards were significantly lower and their inclusion would obfuscate the distinction between separate runs.

Recall from section 5.1.1 that the goal of a leader in this environment is to reach the target landmark, while cooperative adversaries have to prevent the leader from getting close to the landmark while still remaining close to the target themselves. The first plot in figure 5.13 shows reward curves for leaders where the step schedule outperforms other α update schedules. Though the step update performs well for leader policies it struggles for both follower policies. Similarly, the log update shows similar patterns to the step update where it performs relatively well for leader policies but struggles for follower policies. The linear update contains large variance throughout training and negative rewards across all policies as the previously mentioned schedules. The exp schedule seems to perform well for leader policies but its follower policies average to zero reward, indicating that the followers do not learn to exploit the leader's first move. Finally, the sigmoid α update schedule performs fairly poorly with respect to leader policies, but its follower policies indicate that one follower learns to stay close to the target landmark while another follower tries to push the leader away. This is illustrated by follower 2 obtaining a positive reward, while follower 1 receiving lower reward with higher variance.

A comparison between SMARL with $\alpha = 0.75$, sigmoid update and the MADDPG baseline is presented in figure 5.14. As mentioned earlier, the static α runs did not perform well as illustrated by the $\alpha = 0.75$ average reward curves. This is apparent due to the extremes between leader and follower reward curves. The leader performed very poorly

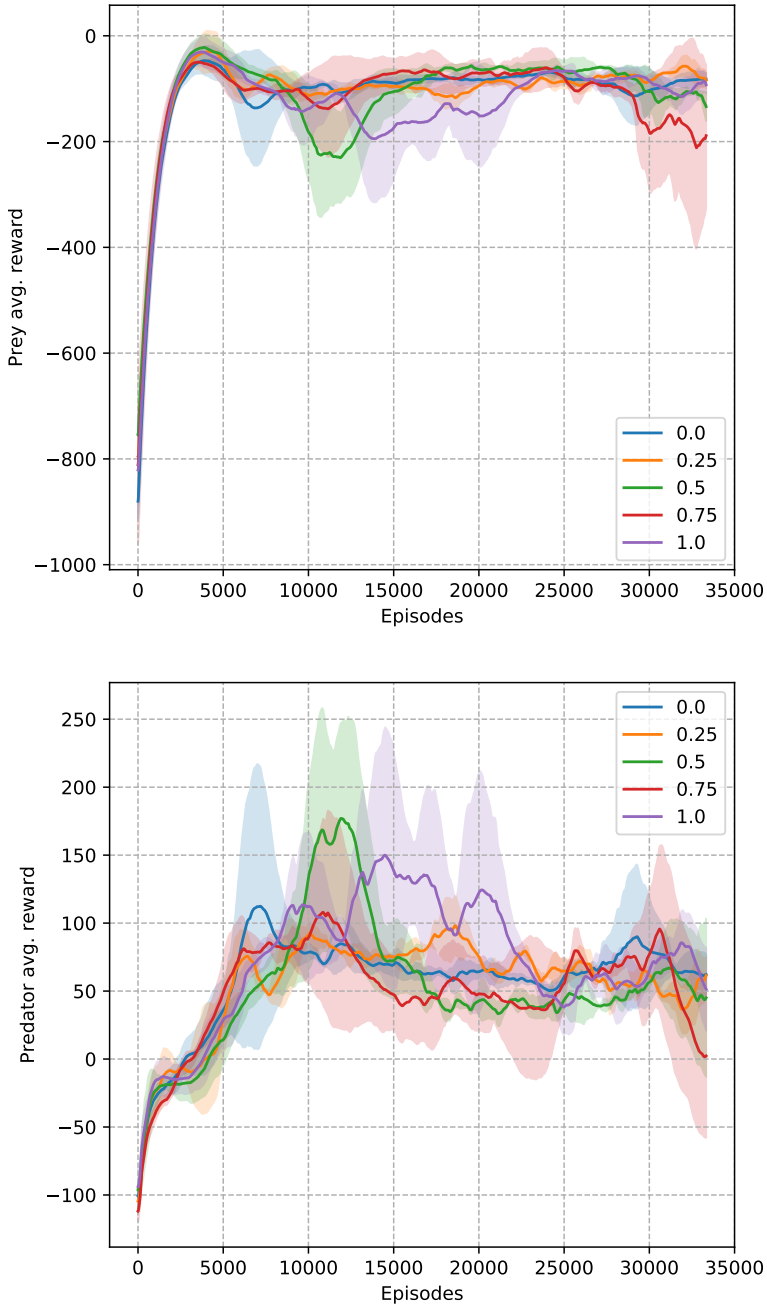

Figure 5.10: Predator Prey average rewards using SMARL with static α values

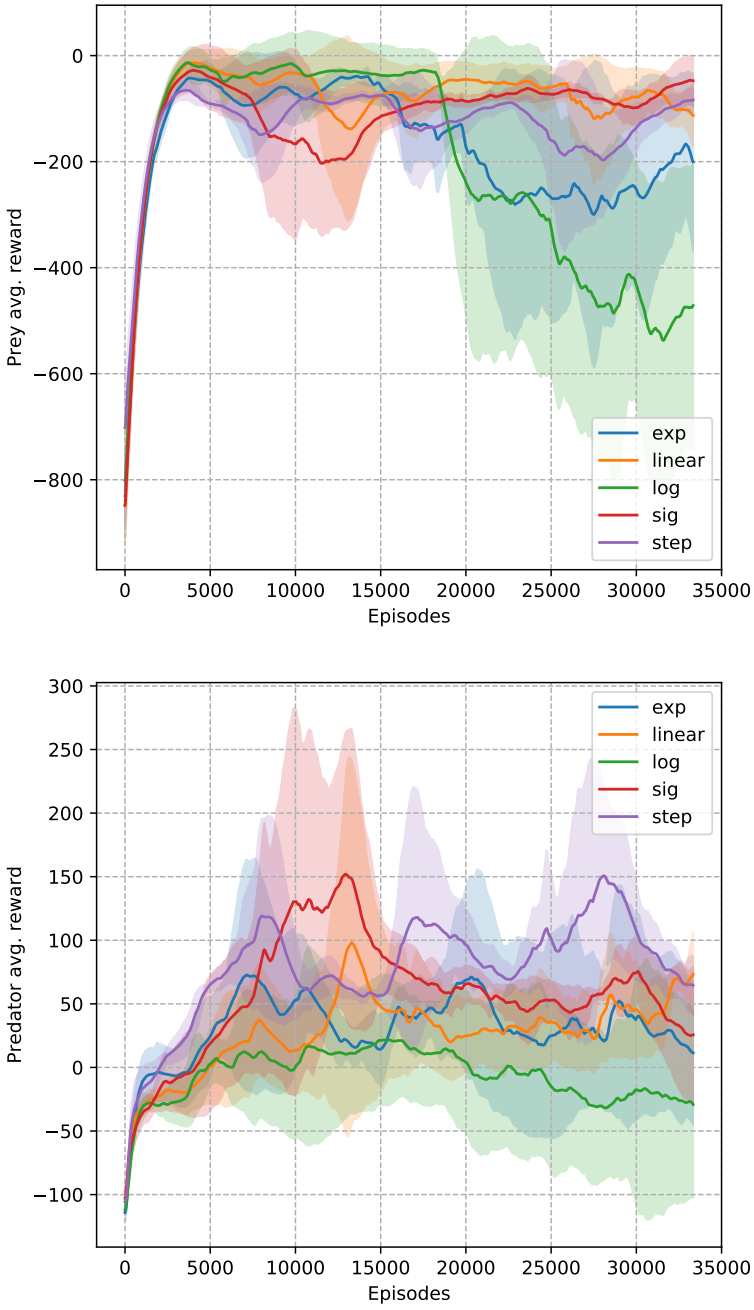

Figure 5.11: Predator Prey average rewards using SMARL with variable α values

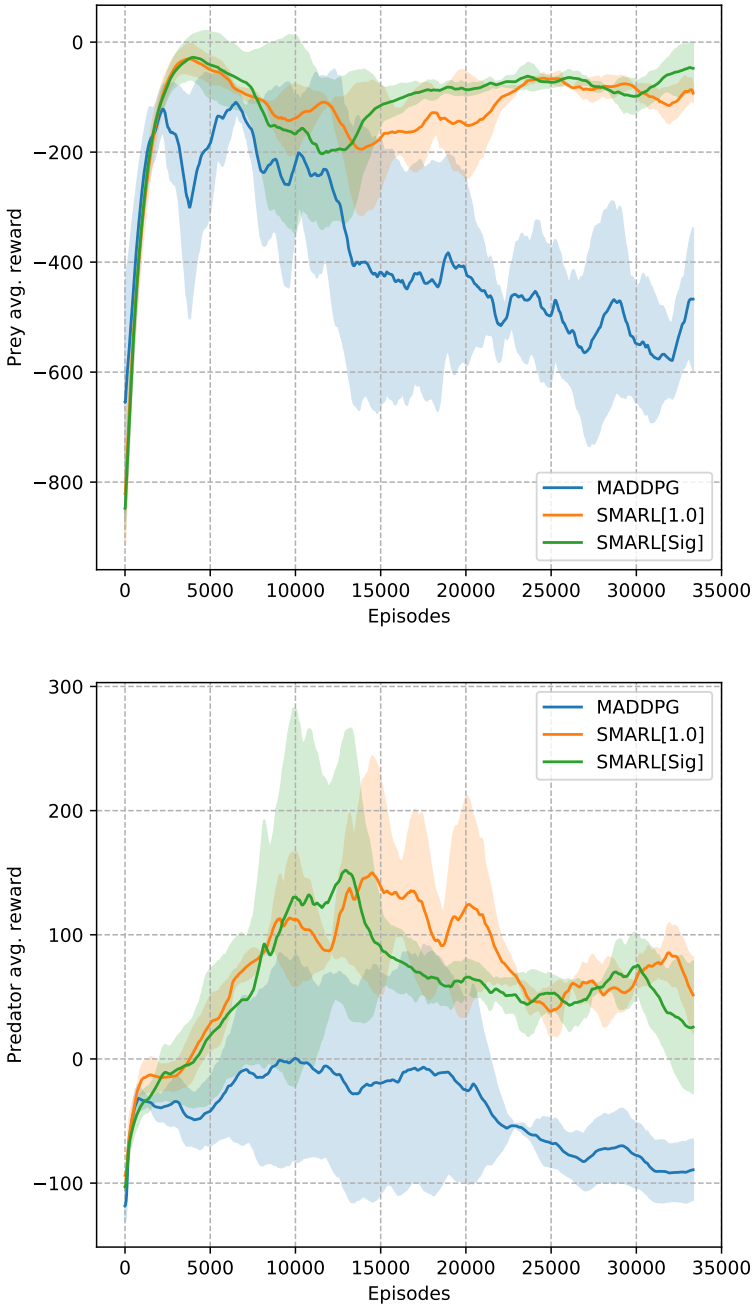

Figure 5.12: Predator Prey Baseline vs. SMARL static α vs. SMARL variable α

while both followers obtained high positive rewards. This indicates that the leader clearly diverged and did not learn a strategy to get closer to the target landmark, while followers learned to remain near the target and were never challenged. In contrast, SMARL with sigmoid α updates was able to learn follower policies which took advantage of knowing the leader's action. Follower 2 in the sig case learned to remain near the target, while follower 1 ensured the leader was kept away. This is visible in our reward curves as follower 1 needed to move away from the target landmark, resulting in lower reward with high variance. The leader was unable to get close to the target since follower 2 maintained its position near this landmark. The baseline performed very well with respect to leader policies, and maintained similar rewards between the followers. This indicates that both followers took initiative in trying to keep the leader away from the target, but this created an opening in the defence which allowed the leader to get closer to the target. This behaviour is understandable since the baseline followers did not have knowledge about the leader's action.

Overall, it is hard to determine which algorithm is superior in this environment. The baseline trains better attackers, while SMARL trains better defenders. This distinction results from the hierarchical assumption used as our basis for SMARL. Our proposed algorithm trains better follower policies in this environment because followers have knowledge about the leader's action and the ability to exploit this information by relying on their cooperative partner. This is not the case in other environments where leaders are not forced into a bottleneck to obtain rewards. This environment highlights the consequences of using a hierarchical assumption in an environment that was not designed for such cases. The use of our assumption in this particle environment resulted in one group learning policies which exploited their position instead of learning general policies which attempt to solve the problem as illustrated by the baseline.

5.3 Summary

In this chapter we presented experimental environments used to test the performance of SMARL along with the results obtained by leader and follower policies in each environment. Section 5.1.2 described our custom built highway driving environment used to test position based hierarchies, while section 5.1.1 described the predator-prey and keep-away multi-agent particle environments. The network architecture and timing analysis for SMARL and the MADDPG baseline is presented in section 5.2. Highway driving average reward curves, in section 5.2.1, for the easy and hard scenarios show that SMARL leader-follower policies with variable α update schedules outperform MADDPG in both scenarios. Particle environment results are presented in section 5.2.2 where SMARL is shown to outperform

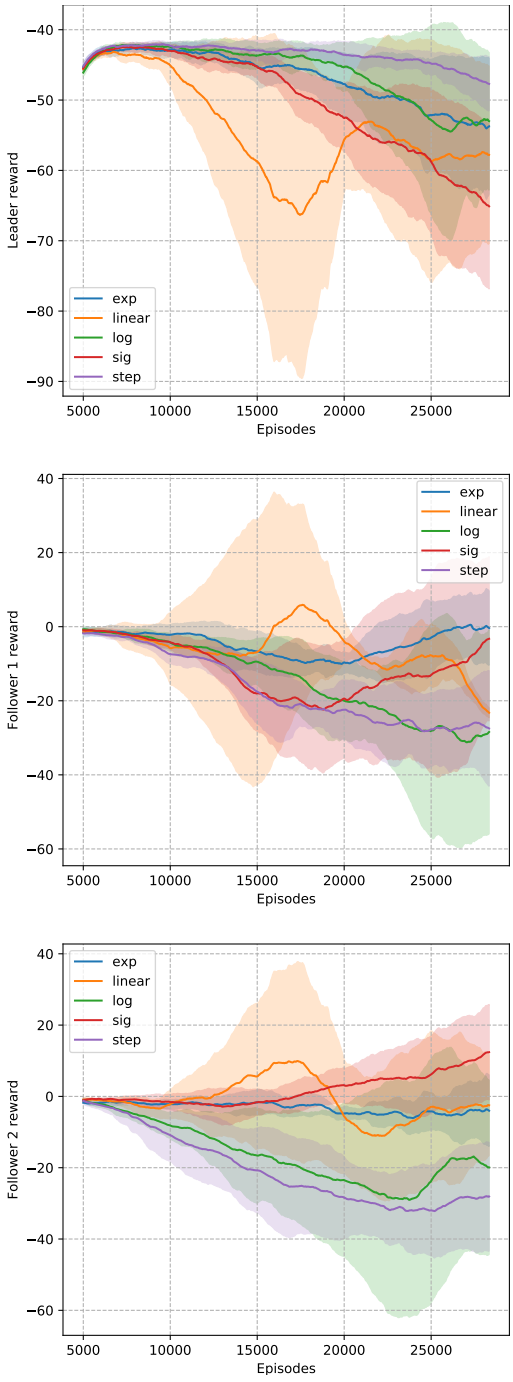

Figure 5.13: Keep Away average rewards using SMARL with variable α values

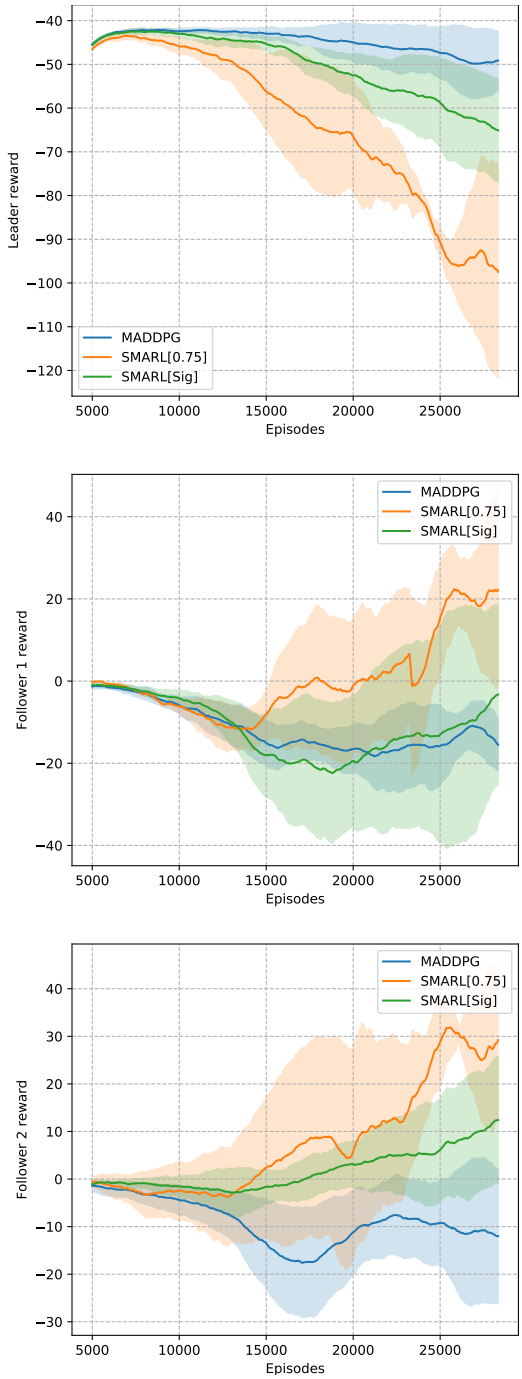

Figure 5.14: Keep Away Baseline vs. SMARL static α vs. SMARL variable α

the baseline for both leader and follower policies. The keep-away particle environment shows that using the hierarchical assumption, exploited by SMARL, can lead to very specialized policies that are not indicative of the desired learning objective. Overall, this chapter highlighted the benefits of using hierarchies within an environment to learn good policies through the use of our SMARL algorithm while also being conservative with its application to the general class of multi-agent environments.

Chapter 6

Conclusions & Future Directions

6.1 Conclusion

In this thesis, we introduced a new method for training policies in multi-agent domains containing hierarchies among agents. Our method integrated the concept of leaders and followers defined in the Stackelberg game theoretic model into reinforcement learning. We leveraged the idea of centralized learning with decentralized execution to train policies in environments with multiple agents. This involved formulating learning objectives which took advantage of hierarchies within an environment while working with continuous state and action spaces. In an effort to work with a larger number of follower agents, we proposed methods to control the impact of follower actions on leader policy updates. This resulted in two variants of our algorithm, one where the influence of follower actions on leader policies was fixed, and another where this influence increases over the course of training. Our algorithm containing gradual increase in follower influence outperformed one with static and rapidly increasing follower influence. We tested our method using multi-agent particle environments and our custom built highway driving environment. The results show that our method outperformed the baseline with respect to all agents in most domains by taking advantage of leader-follower dynamics within the environment.

6.2 Future Directions

As our work is based on policy gradient methods, it benefits from the use of domains with continuous state and action space. It would be very beneficial to reproduce this work

with discrete action domains, since existing methods rely on searching the full action space [15]. This would involve developing a method to work with non-deterministic policies that contain a softmax output unit. Our algorithm also benefited from the use of a variable hyper-parameter α, used for controlling follower influence on leader policies. This α can be dynamically updated during training, by taking into account leader's policy performance, instead of following a fixed schedule, to make better use the follower influence.

Another direction of interest is studying how the Stackelberg model can be applied to the training process introduced in Learning with Opponent-Learning Awareness (LOLA) [7]. LOLA allowed each agent to shape the anticipated learning of its opponents, but optimized its expected return using opponent parameters which were updated using a naive learning step. This optimization can be improved for hierarchical environments by using leader actions and follower responses to update opponent parameters in an informed manner instead of relying on naive learning steps.

Our proposed algorithm used the Q-value for actor-critic training, but it could benefit from the use of an advantage function given a good baseline. This advantage function can be computed using a counterfactual baseline described in Counterfactual multi-agent policy gradients [8]. These counterfactual baselines could help address the multi-agent credit assignment problem faced in the particle environments where followers need to cooperate and share a global reward.

We hope our proposed approach to learning policies in multi-agent hierarchical environments can lead to development of new multi agent reinforcement learning algorithms which are inspired by game theory and applicable to real world multi agent problems.

References

[1] Kareem Amin, Satinder Singh, and Michael P Wellman. Gradient methods for stackelberg security games. In *Conference on Uncertainty in Artificial Intelligence*, pages 2–11, 2016.

[2] Simon P Anderson and Maxim Engers. Stackelberg versus cournot oligopoly equilibrium. *International Journal of Industrial Organization*, 10(1):127–135, 1992.

[3] Richard Bellman. The theory of dynamic programming. Technical report, Rand corp santa monica ca, 1954.

[4] Yu-Han Chang, Tracey Ho, and Leslie P Kaelbling. All learning is local: Multi-agent learning in global reward games. In *Advances in neural information processing systems*, pages 807–814, 2004.

[5] Chi Cheng, Zhangqing Zhu, Bo Xin, and Chunlin Chen. A multi-agent reinforcement learning algorithm based on stackelberg game. In *2017 6th Data Driven Control and Learning Systems (DDCLS)*, pages 727–732. IEEE, 2017.

[6] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

[7] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[8] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[10] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[11] Majed Haddad, Zwi Altman, Salah Eddine Elayoubi, and Eitan Altman. A nash-stackelberg fuzzy q-learning decision approach in heterogeneous cognitive networks. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6. IEEE, 2010.

[12] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, Dj Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pages 3040–3049, 2019.

[13] Donald Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

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

[15] Ville Könönen. Asymmetric multiagent reinforcement learning. *Web Intelligence and Agent Systems: An international journal*, 2(2):105–121, 2004.

[16] Ville Könönen. Dynamic pricing based on asymmetric multiagent reinforcement learning. *International journal of intelligent systems*, 21(1):73–98, 2006.

[17] Leslie Lamport. *LaTeX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[18] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[19] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[20] Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, and Stuart Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] Alexander Liniger and John Lygeros. A noncooperative game approach to autonomous racing. *IEEE Transactions on Control Systems Technology*, 2019.

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

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

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

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

[27] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2, 2017.

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

[30] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[31] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[32] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[33] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[34] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

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

[36] Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems*, pages 871–878, 2004.

[37] Kagan Tumer and Adrian Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 255. ACM, 2007.

[38] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[39] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

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