Reward Shaping in Single and Multi-Agent Deep Reinforcement Learning

Baicen Xiao

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Radha Poovendran, Chair

Sreeram Kannan

Eli Shlizerman

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

Reward Shaping in Single and Multi-Agent
Deep Reinforcement Learning

Baicen Xiao

Chair of the Supervisory Committee:
Radha Poovendran
Electrical and Computer Engineering

Rapid strides made in the development of computing infrastructure over the last ten years have played a crucial role in significantly advancing the state-of-the-art in reinforcement learning. These developments have enabled the successful implementation of reinforcement learning algorithms in complex domains, including computer games, molecular design and robotics. However, it remains difficult for a reinforcement learning agent to learn to effectively complete tasks, especially when the reward provided by the environment is sparse or significantly delayed. In these situations, the agent will not be able to receive immediate feedback on its actions. This is called the *credit assignment problem*. In many scenarios, it is hard to design a reward scheme that is *dense*- i.e., a scheme that provides frequent or timely rewards to the agent at each intermediate time-step. It is therefore critical to develop tools that can guide a reinforcement learning agent towards promising solutions efficiently in environments with sparse reward.

Reward shaping refers to a class of credit assignment methods, which augments the reward from the environment with an additional reward signal, with the goal of giving immediate feedback to the agent. The theme of this thesis is to integrate reward shaping into deep reinforcement learning algorithms to *i)* enhance the speed of learning; *ii)* improve the performance of the learned policies by allowing the agent to obtain higher rewards. In

this thesis, we consider three different types of information that can be utilized to perform reward shaping on deep reinforcement learning algorithms.

First, we focus on information which is in the form of potential functions. The difference between the values of a potential function at any two points is independent of the path taken in traveling from one point to the other. Potential-based shaping advice refers to a class of methods that uses the difference of potential functions as the shaping reward. We develop algorithms to impart a potential-based shaping advice scheme to policy gradient algorithms in both single-agent and multi-agent reinforcement learning.

In some scenarios, the domain knowledge may not be in the form of potential functions. Instead, human operators may be available to provide feedback during training. Therefore, we seek to effectively integrate feedback signals supplied by a human operator with deep reinforcement learning algorithms in high-dimensional state spaces. We propose a framework FRESH (Feedback-based REward SHaping) which is designed to transform human feedback into a shaping reward that can be augmented to the environment reward.

Potential-based advice and human expert, however, may not be available in scenarios where the number of agents increases or the environment is complex. Without prior domain knowledge, we propose an algorithm which can automatically perform effective long-term temporal credit assignment based on the interaction history of agents. In order to solve the temporal credit assignment problem in multi-agent environments with delayed rewards, it is critical to identify the relative importance of: i) each agent's state at any single time-step (*agent dimension*); ii) states along the length of an episode (*temporal dimension*). We introduce *Agent-Temporal Attention for Reward Redistribution in Episodic Multi-Agent Reinforcement Learning (AREL)* to address these challenges.

In each case, our experiments demonstrate that the reward shaping methods that we develop in this thesis help improve the state-of-the-art deep reinforcement learning algorithms to obtain higher average rewards and faster learning speeds.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am incredibly fortunate to have been given the opportunities that have led me to study as a PhD student at Network Security Lab (NSL). This disseration would have not been possible without the support of my collegues, friends and family.

I thank my advisor Prof. Radha Poovendran, for his invaluable guidance and mentoring over the years at University of Washington (UW). He taught me how to do research from identifying the right questions, decomposing a complex problem to simpler sub-problems, to presenting the results. His commitment to both research and excellence has inspired me throughout my PhD. His endless support in both academic work and daily life is one of the most important factors for my success at UW. I would also like to thank the rest of my PhD supervisory committee members, Prof. Sreeram Kannan, Prof. Eli Shlizerman and Prof. Arvind Krishnamurthy for their time and insightful discussion.

I thank the former member of NSL and one of my committee members, Prof. Bhaskar Ramasubramanian, who is my mentor, research collaborator, and also friend. I learned a lot from Bhaskar. He not only gave me advices on acedemic writing and presentation but also backed me up with encouragement. I extend my thanks to all other current or former NSL members, including Dr. Xuhang Ying, Dr. Hossein Hosseini, Dr. Sang Sagong, Joanna Mazer, Dinuka Sahabandu, Kalana Sahabandu, Dr. Yize Chen, Prof. Shana Moothedath, Sarat Bobbili, Shruti Misra, Dr. Phillip Lee and Qiong Lin for their support and friendship. I also want to thank Prof. Hui Liu. Without his support and reference, I would not have obtained the opportunity to study at UW.

I would also like to acknowledge the generous funding support I have received from the following sources: ONR grant N00014-17-12946 for Feedback-Driven Learn to Reason

in Adversarial Environments for Autonomic Cyber Systems (L2RAVE) and a grant from Boeing for Cyber Security Machine Learning.

At last, I extend my deepest gratitude to my mother, father, and my girlfriend for their priceless love and support.

DEDICATION

to my parents and girlfriend

whose love and support made this thesis possible

Chapter 1

INTRODUCTION

Reinforcement learning (RL) is a framework that enables an agent to explore an environment in order to maximize its expected long-term reward, where the reward signal is supplied by the environment. A major attraction of this approach is that the agent can learn to complete tasks even when a model of the environment (i.e. transition probabilities between states) is unknown. With the dramatic progress in the development of computational resources, model-free RL algorithms have been successfully applied to many domains, including computer games [61, 75] and robotics [32, 50]. Although RL algorithms have shown impressive results, they can struggle in certain environments. This is especially true if the reward signal provided by the environment is sparse or delayed.

Figure 1.1: Atari game Skiing. The objective of the player (i.e., the agent) is to reach the bottom of the slope as quickly as possible. The player obtains a reward in the form of a time-bonus for passing through gates, and is penalized for slowing down (e.g., steering) or hitting obstacles (e.g., trees or mud).

Environments with sparse or delayed reward are very common. For example, Figure 1.1 shows the Atari game Skiing. The Skiing environment consists of gates, trees and mud. The

task of the player (i.e., agent) is to learn to reach the end of the course as rapidly as possible. The player is also expected to pass through gates, and avoid trees and mud. Not satisfying these objectives will result in a time-penalty to the agent. The agent's speed is affected by the actions it chooses and whether it hits trees or mud. In this environment, it can be difficult to precisely gauge if actions of the agent before the game ends are good or bad. For example, missing some gates could be reasonable in order to achieve higher speed. This makes the design of a dense reward signal that assigns a reward to actions taken at every time-step difficult. On the other hand, it will be easier to specify a 'sparse' reward signal that gives the agent a summary reward only when it reaches the end of the course (and zero reward at all other time). This has the advantage of specifying the task objective accurately. However, a shortcoming is that the agent will not receive immediate feedback on the quality of its actions before the end of the game.

Reward shaping is one way to provide immediate feedback on actions so that the original sparse reward signal becomes more dense. Our goal is to improve deep reinforcement learning via incorporating reward shaping techniques in environments with delayed reward. Specifically, we study three forms of information to perform reward shaping in single- and multi-agent RL: 1) potential functions; 2) human feedback and 3) interaction history of agents.

1.1 Summary of Research Contributions

It may be possible to guide an RL agent towards more promising solutions faster, if it is equipped with some form of knowledge about the environment. This knowledge can be encoded by modifying the reward signal received by the agent during training. However, the modification must be carried out in a principled manner, since providing an additional reward at each step might distract the agent from the true goal such that the agent tries to maximize the additional reward instead of the environment reward. Potential-based reward shaping (PBRS) [64] and potential-based advice (PBA) [94] are principled methods that augment the reward in an environment specified by a Markov Decision Process (MDP) with

a term that is a difference of potentials. We study the addition of PBRS and PBA to environment rewards in settings where the optimal policy will be stochastic, and state and action spaces are continuous. We additionally provide guarantees on the convergence of an advantage actor-critic architecture that is augmented with a PBA scheme. In Chapter 3, we describe the proposed methods in detail, which appear in [100].

We then extend PBRS and PBA to multi-agent reinforcement learning (MARL). In MARL, any single agent interacts not only with the environment, but also with other agents. Since actions of an agent will affect the behavior of others, the environment becomes non-stationary from the perspective of any single agent, which means that guarantees of standard RL methods will be lost. We propose a multi-agent actor-critic method called Shaping Advice in Multi-agent deep reinforcement learning (SAM) to address this challenge. SAM uses a centralized critic to estimate value functions for each agent, and decentralized actors to optimize agent policies. Decentralized actors make it possible for each agent to use its own observations to determine its actions. Decentralization of actors is more practical when agents may not be able to communicate with each other or the number of agents is large. SAM exploits information about the environment and task to define shaping advice for each agent. SAM can be applied to cooperative or competitive multi-agent frameworks. We prove that convergence of policy gradients and value functions when using SAM implies convergence of these quantities in same environment in the absence of SAM. The details of SAM are presented in Chapter 4.

In some scenarios, the domain knowledge may not be in the form of potential functions. However, human operators may be available to provide feedback during training. Although RL algorithms have been observed to perform impressively in certain environments, humans are usually more efficient in terms of the number of actions required to obtain higher rewards [34]. This has especially been observed in environments with high-dimensional state spaces, like video games, where states are raw pixels of images or snapshots of videos [67]. In these settings, any prior knowledge that a human might have about the environment, and their ability to learn from the environment, are crucial in determining success. We seek

to effectively integrate human feedback with deep RL algorithms in high-dimensional state spaces. We term this *FRESH*, for Feedback-based REward SHaping. In FRESH, a human operator provides feedback on actions taken by the RL agent. During training, the operator is presented with trajectories (sequences of states and actions) from a replay buffer and indicates whether an action at a state in the trajectory is good or bad. This human feedback will be summarized by neural networks into a shaping reward, which will be combined with the environment reward to train the agent. We evaluate FRESH on the Bowling and Skiing Atari games in the Arcade Learning Environment and observe that FRESH is able to outperform state-of-the-art DRL algorithms in both environments. In Chapter 5, we describe the FRESH framework, which appears in [99].

In the last topic of this thesis, we aim to scale up reward shaping to settings with multiple agents, where neither domain knowledge in terms of potential functions, nor human operators may be available. Specifically, we consider multi-agent reinforcement learning (MARL) tasks where agents receive a shared global reward at the end of an episode. We propose an algorithm which can automatically perform effective long-term temporal credit assignment based on the interaction history of agents. In order to solve the temporal credit assignment problem in multi-agent environments with episodic rewards, it is critical to identify the relative importance of: i) each agent's state at any single time-step (*agent dimension*); ii) states along the length of an episode (*temporal dimension*). We introduce *Agent-Temporal Attention for Reward Redistribution in Episodic Multi-Agent Reinforcement Learning (AREL)* to address these two challenges. AREL uses attention mechanisms to characterize the influence of actions on state transitions along trajectories (*temporal attention*), and how each agent is affected by other agents at each time-step (*agent attention*). The redistributed rewards predicted by AREL are dense, and can be integrated with any given MARL algorithm. We evaluate AREL on challenging tasks from the Particle World environment and the StarCraft Multi-Agent Challenge. AREL results in higher rewards in Particle World, and improved win rates in StarCraft compared to three state-of-the-art reward redistribution methods.

1.2 Related Work

Shaping or augmenting the reward received by an RL agent in order to enable it to learn optimal policies faster has been an active area of research. A curiosity-based RL algorithm for sparse reward environments was presented in [67], where an intrinsic reward signal characterized the prediction error of the agent as a curiosity reward. The reward received by the agent was augmented with a function that represented the number of times the agent had visited a state in [82]. Entropy regularization as a way to encourage exploration of policies during the early stages of learning was studied in [62] and [95]. This was used to lead a policy towards states with a high reward in [47] and [48].

Static potential-based functions were shown to preserve the optimality of deterministic policies in [64]. This property was extended to dynamic potential-based functions in [17]. The authors of [93] showed that when an agent learned a policy using Q-learning, applying PBRS at each training step was equivalent to initializing the Q-function with the potentials. They studied value-based methods, but restricted their focus to learning deterministic policies. The authors of [33] demonstrated a method to transform a reward function into a potential-based function during training. The potential function in PBA was obtained using an 'experience filter' in [49]. Shaping advice in the form of potential functions was used for MARL in [16]. This was empirically shown to accelerate learning of discrete policies in a discrete action multi-agent system. Heuristic functions derived from a deterministic preference policy were used as advice to accelerate learning in two-player discrete stochastic games in [5].

The credit assignment problem has been studied for cooperative MARL in recent work. The CTDE framework was used with heuristic-based shaping rewards that were a function of individual agent states in [29]. The authors of [77] proposed value decomposition networks that decomposed a centralized value into a sum of individual agent values to determine each one's contributions. An additional assumption on monotnonicity of the centralized value function was imposed in QMIX [70] to assign credit to an agent. [102] proposed Q-value path decomposition to decompose global Q-values along trajectory paths to assign credit

for agents. Intrinsic rewards were used to define a proxy critic for each agent in [19]. This was used to transform the MARL objective to a bi-level optimization problem. Attention mechanisms have been used for multi-agent credit assignment in recent work. The authors of [58] used an attention mechanism with a CTDE-based algorithm to enable each agent effectively model policies of other agents (from its own perspective). Hierarchical graph attention networks proposed in [72] modeled hierarchical relationships among agents and used two attention networks to effectively represent individual and group level interactions. The authors of [37, 53] combined attention networks with graph-based representations to indicate the presence and importance of interactions between any two agents. The above approaches used attention mechanisms primarily to identify relationships between agents at a specific time-step. They did not consider long-term temporal dependencies, and therefore may not be sufficient to learn policies effectively when rewards are delayed.

The role of feedback provided by a human to an agent in RL settings has also been a popular topic of active research. A survey of recent research in using human guidance for deep RL tasks was presented in [105]. A framework called TAMER (Training an Agent Manually via Evaluative Reinforcement) that enabled *shaping* (interactively training an agent via an external signal provided by a human) was presented in [41]. This work was extended to enable human feedback to augment an RL agent that learned using an MDP reward signal in [42, 43]. The outcome was that using this human feedback achieved a significant reduction in sample complexity. More recently, the authors of [91] proposed Deep-TAMER, an architecture that extends the TAMER framework to environments with high-dimensional state spaces. In Deep-TAMER, the policy was given by a neural network that was trained via supervised learning using data from feedback provided by a human operator (while not considering the reward given by the environment). Another extension of TAMER, DQN-TAMER [1], modeled additional characteristics of human observers like inferring human reward from facial expressions. Signals provided by the human operator in TAMER was a numerical value indicative of how good the agent's behavior was, in the opinion of the operator.

Preference-based RL [13] was used to communicate complex goals to allow systems to interact with real-world environments in a meaningful way. Although this approach required a human observer to compare trajectories and provide feedback during training, it alleviated the need for expert observers, since non-experts can easily compare and distinguish between 'good' and 'bad' trajectories. The human preferences were translated into a scalar reward, which was then used as a reward signal to an RL algorithm. This allowed the RL agent to directly learn from expert preferences. However, this approach is limited by assumptions on the existence of a (total) order among the set of trajectories. A survey of preference-based RL methods was presented in [96] for the interested reader.

Another policy-based method to provide human feedback is called *policy shaping* [28]. In this method, feedback was a label on the optimality of an action, rather than a reward signal added to the reward from the environment. In this setup, the RL agent, in addition to receiving a reward from the environment, obtained an indication of whether the most recent action was correct or incorrect. This label was used to infer the human's belief of the optimal policy in the current state. Extensions to this work considered the effect of human attention [22] and settings where the robot could request feedback in states where it was not sure of [39]. A similar approach that studied the interpretation of feedback strategies adopted by a human trainer as a probabilistic model was presented in [54], and this resulted in a strategy-aware Bayesian learning (SABL) algorithm.

Demonstrations provided by a human operator were used to synthesize a 'baseline policy' in Human-Agent Transfer (HAT) [83]. This baseline policy was then used to guide learning during the RL procedure. CHAT [89] extended HAT to consider possible errors made while summarizing demonstrations, and used this uncertainty to improve performance. As an alternative to providing demonstrations, the authors of [71] presented DAGGER, an iterative imitation learning method which required a domain expert be available to provide correct actions during the entire learning process. A subsequent paper [38] presented HG-DAGGER that predicted the performance of the agent using a threshold that modeled uncertainty.

When prior knowledge of the problem domain is not available, recent work has studied

temporal credit assignment in single-agent RL with delayed rewards. An approach named RUDDER [2] used contribution analysis to decompose episodic rewards by computing the difference between predicted returns at successive time-steps. Parallelly, the authors of [52] proposed using natural language processing models for carrying out temporal credit assignment for episodic rewards. The scalability of the above methods to MARL, though, can be a challenge due to the exponential growth in the size of the joint observation space [55]. A method for temporal redistribution of episodic rewards in single and multi-agent RL was recently presented in [26]. A 'surrogate objective' was used to uniformly redistribute an episodic reward along a trajectory. However, this work did not use information from sample trajectories to characterize the relative contributions of agents at intermediate time-steps along an episode.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 presents required preliminaries to the classic RL formulation and potential-based reward shaping. Chapter 3 presents our results on applying potential-based advice to learning stochastic policies in single-agent settings. Potential-based shaping methods for multi-agent scenarios are reported in Chapter 4. Reward shaping based on human feedback is given in Chapter 5. Agent-temporal attention method is introduced in 6. We conclude this thesis in Chapter 7.

Chapter 2

PRELIMINARIES

2.1 Single and Multi-Agent Reinforcement Learning

In the single-agent setting, a reinforcement learning problem is generally modeled as an Markov Decision Process (MDP) which is a tuple $(S, A, \mathbb{T}, \rho_0, R, \gamma)$ [68]. S is the set of states, A the set of actions, $\mathbb{T} : S \times A \times S \to [0, 1]$ encodes $\mathbb{P}(s_{t+1}|s_t, a_t)$, the probability of transition to s_{t+1}, given current state s_t and action a_t. ρ_0 is a distribution over the initial states, and $\gamma \in (0, 1]$ is a discounting factor. $R : S \times A \to \mathbb{R}$ denotes the reward that the agent receives when transitioning from s_t while taking action a_t. In this thesis, $|R| < \infty$. Similarly, multi-agent RL is modeled as a *stochastic game*. When there are n players, then the stochastic game can be denoted as a tuple $\mathcal{G} = (S, A^1, \ldots, A^n, P, R^1, \ldots, R^n, O^1, \ldots, O^n, \rho_0, \gamma)$. S is the set of states, A^i is the action set of player i, $P : S \times A^1 \times \cdots \times A^n \times S \to [0, 1]$ encodes $\mathbb{P}(s_{t+1}|s_t, a_t^1, \ldots, a_t^n)$, the probability of transition to state s_{t+1} from s_t, given the respective player actions. $R^i : S \times A^1 \times \cdots \times A^n \to \mathbb{R}$ is the reward obtained by agent i when transiting from s_t while each player takes action a_t^i. O^i is the set of observations for agent i. At every state, each agent receives an observation correlated with the state: $o^i : S \to O^i$.

In single-agent RL, the goal for an agent [78] is to learn a *policy* π, in order to maximize the expected discounted cumulative reward, $J := \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. Here, the expectation is taken over the trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$ induced by policy π. If $\pi : S \to A$, the policy is *deterministic*. On the other hand, a randomized policy returns a probability distribution over the set of actions, and is denoted $\pi : S \times A \to [0, 1]$. For multi-agent RL, the goal for each agent is generally to maximize its own cumulative reward [85]. We can define the policy for each agent similar to single-agent cases. A stochastic policy for agent i is a map $\pi^i : O^i \times A^i \to [0, 1]$.

The value of a state-action pair (s, a) following policy π is represented by the *Q-function*, written as $Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)|s_0 = s, a_0 = a]$. The Q-function allows us to calculate the state value $V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a)]$. The advantage of a particular action a, over other actions at a state s is defined by $A^\pi(s,a) := Q^\pi(s,a) - V^\pi(s)$. In multi-agent cases, let $\boldsymbol{\pi} := \{\pi^1, \ldots, \pi^n\}$ and $s := (s^1, \ldots, s^n)$. Following [55], in the simplest case, $s^i = o^i$ for each agent i. The Q-function is defined as $Q_i^{\boldsymbol{\pi}}(s, a^1, \ldots, a^n) := \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^t R^i(s_t, \boldsymbol{a}_t)|s_0 = s, a_0^1 = a^1, \ldots, a_0^n = a^n]$. Similarly, the state value function $V_i^{\boldsymbol{\pi}}(s) = \mathbb{E}_{\{a^i\sim\pi^i\}_{i=1}^n}[Q_i^{\boldsymbol{\pi}}(s, a^1, \ldots, a^n)]$.

2.2 Value-based and Policy-based Methods

The RL problem has two general solution techniques. *Value-based* methods determine an optimal policy by maintaining a set of reward estimates when following a particular policy. At each state, an action that achieves the highest (expected) reward is taken. Typical value-based methods to learn greedy (determininistic) policies include Q-learning and Sarsa-learning [78]. Recently, the authors of [30] proposed *soft Q-learning*, which is a value-based method that is able to learn stochastic policies.

In comparison, *policy-based* methods directly search over the policy space [78]. Starting from an initial policy, specified by a set of parameters, these methods compute the expected reward for this policy, and update the parameter set according to certain rules to improve the policy. Policy gradient [79] is one way to achieve policy improvement. This method repeatedly computes (an estimate of) the gradient of the expected reward with respect to the policy parameters. Policy-based approaches usually exhibit better convergence properties, and can be used in continuous action spaces [31]. They can also be used to learn stochastic policies. REINFORCE and actor-critic are examples of policy gradient algorithms [78].

In the following, we introduce some notations of policy gradient methods for multi-agent senarios, which will be used in Chapter 4. The policy π^i for agent i is parameterized by θ_i. We denote this by π_{θ_i}, and define $\boldsymbol{\pi_\theta} := \{\pi_{\theta_1}, \ldots, \pi_{\theta_n}\}$. We assume that $\pi_{\theta_i}(a^i|o_i) > 0$ for all θ_i and is continuously differentiable with respect to θ_i. The value of the parameterized policy $\boldsymbol{\pi_\theta}$ for agent i is then: $J_i(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{\pi_\theta}}[\sum_t \gamma^t R_t^i]$. We use $-i$ to denote all agents

other than agent i. We define the *accumulated return* for agent i from time t onwards as $G_i(s_t, a_t^i, a_t^{-i}) := \sum_{j=t}^{\infty} \gamma^{j-t} R_j^i$. Then, from the policy gradient theorem [78]:

$$\nabla_{\theta_i} J_i(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\pi_\theta}}[\nabla_{\theta_i} \log\ \pi_{\theta_i}(a_t^i|o_t^i)\ G_i(s_t, a_t^i, a_t^{-i})] \tag{2.1}$$

This form of policy gradient is called REINFORCE. In actor-critic methods [44, 78], the *critic* estimates a value function. The *actor* (policy) is learned by following a gradient that depends on the critic. In actor-critic, the accumulated return in equation 2.1 can be replaced by $Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i}) - b_i(s_t)$, where $b_i(s_t)$ is a baseline used to reduce the variance. When $b_i(s_t) = V_i^{\boldsymbol{\pi_\theta}}(s_t)$, this difference is called the *advantage*.

The authors of [55] extended multi-agent actor-critic to work with deterministic policies, and termed their approach multi-agent deep deterministic policy gradient (MADDPG). Formally, consider a game with n players whose policies are parameterized by $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_n\}$. Let $\boldsymbol{\mu} := \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_n\}$ be the set of the agents' policies. Then, in MADDPG, the gradient of the expected return for agent i following policy $\boldsymbol{\mu}_i$ is:

$$\nabla_{\theta_i} J_i(\boldsymbol{\theta}) = \mathbb{E}_{s,a\sim\mathcal{B}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(o^i) \nabla_{a^i} Q_i^{\boldsymbol{\mu}}(s, a^1, \ldots, a^n)|_{a^i=\boldsymbol{\mu}_i(o^i)}] \tag{2.2}$$

In the above equation, $Q_i^{\boldsymbol{\mu}}(s, a^1, \ldots, a^n)$ is a centralized action-value function for agent i. In the simplest case, the state information $s = [o^1, \ldots, o^n]$, but additional information can be included during training.

2.3 PBRS and PBA

Reward shaping methods augment the environment reward R with an additional reward $F \in \mathbb{R}$, and $|F| < \infty$. This changes the structure of the original MDP $M(= (S, A, \mathbb{T}, \rho_0, R))$ to $M' = (S, A, \mathbb{T}, \rho_0, R + F)$. The goal is to choose F so that an optimal policy for M', $\pi^*_{M'}$, is also optimal for the original MDP M. *Potential-based reward shaping* (PBRS) schemes were shown to be able to preserve the optimality of deterministic policies in [64].

In PBRS, the function F is defined as a difference of *potentials*, $\phi(\cdot)$. Specifically, $F(s_t, a_t, s_{t+1}) := \gamma\phi(s_{t+1}) - \phi(s_t)$. Then, the Q-function, $Q^*_M(s, a)$, of the optimal greedy

policy for M and the optimal Q-function $Q^*_{M'}(s,a)$ for M' are related by: $Q^*_{M'}(s,a) = Q^*_M(s,a) - \phi(s)$. Therefore, the optimal greedy policy is not changed [64, 17], since:

$$\begin{aligned}\pi^*_{M'}(s) &\in \arg\max_{a\in A}\ Q^*_{M'}(s,a) \\ &= \arg\max_{a\in A}\ \big(Q^*_M(s,a) - \phi(s)\big) = \arg\max_{a\in A}\ Q^*_M(s,a).\end{aligned}$$

The authors of [94] augmented $\phi(s)$ to include action a as an argument. They termed this *potential-based advice* (PBA). There are two forms– *look-ahead PBA* and *look-back PBA*– respectively defined by:

$$F(s_t, a_t, s_{t+1}, a_{t+1}) = \gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t) \tag{2.3}$$

$$F(s_t, a_t, s_{t-1}, a_{t-1}) = \phi(s_t, a_t) - \gamma^{-1}\phi(s_{t-1}, a_{t-1}). \tag{2.4}$$

For the look-ahead PBA scheme, the state-action value function for M following policy π is given by:

$$Q^\pi_M(s,a) = Q^\pi_{M'}(s,a) + \phi(s,a). \tag{2.5}$$

The optimal greedy policy for M can be recovered from the optimal state-action value function for M' from:

$$\pi^*_M(s_t) \in \arg\max_{a\in A}\big(Q^*_{M'}(s_t,a) + \phi(s_t,a)\big). \tag{2.6}$$

The optimal greedy policy for M using look-back PBA can be recovered similarly.

In the multi-agent case, the potential-based *shaping advice* for an agent at each time is a function of states and actions of all agents. Similarly, the shaping advice can take one of two forms, *look-ahead* and *look-back*, respectively given by:

$$F^i_t(s_t, a^i_t, a^{-i}_t, s_{t+1}, a^i_{t+1}, a^{-i}_{t+1}) := \gamma\phi_i(s_{t+1}, a^i_{t+1}, a^{-i}_{t+1}) - \phi_i(s_t, a^i_t, a^{-i}_t) \tag{2.7}$$

$$F^i_t(s_t, a^i_t, a^{-i}_t, s_{t-1}, a^i_{t-1}, a^{-i}_{t-1}) := \phi_i(s_t, a^i_t, a^{-i}_t) - \gamma^{-1}\phi_i(s_{t-1}, a^i_{t-1}, a^{-i}_{t-1}) \tag{2.8}$$

We will denote by $\mathcal{G}'$ the n player stochastic game that is identical to $\mathcal{G}$, but with rewards $R'^i := R^i + F^i$ for each i.

The shaping advice is a heuristic that uses knowledge of the environment (e.g. geometrical characteristics) and the task, along with information available to the agent. For example, in the particle world tasks that we study, each agent has access to positions of other agents and those of landmarks, relative to itself. This is used to give feedback on the quality of actions taken by the agent at each time step.

Chapter 3

POTENTIAL-BASED ADVICE FOR STOCHASTIC POLICY LEARNING

3.1 Motivation

In RL, the agent 'learns' to complete a task by maximizing the reward signal supplied by the environment. Although RL algorithms have been successfully applied in many fields, including robotics [32, 50] and games [61, 75], it remains difficult for an RL agent to master new tasks in unseen environments when the reward given by the environment is sparse.

If some form of *prior knowledge* about the environment is available, an RL agent may be trained more efficiently towards promising solutions. The prior knowledge can be encoded by modifying the reward signal received by the agent during training. However, the modification must be carried out in a principled manner, such that the optimal policy will not be changed or can be easily recovered [69]. Potential-based reward shaping (PBRS) is one such method that augments the reward in an environment specified by a Markov Decision Process (MDP) with a term that is a difference of *potentials* [64]. Potential functions in PBRS are typically functions of states. This could be a limitation, and to allow for imparting more information to the agent, a potential-based advice (PBA) scheme was proposed in [94]. The potential functions in PBA include both states and actions as their arguments.

To the best of our knowledge, PBRS and PBA schemes in the literature [17, 64, 94] assume that an optimal policy is deterministic. This will not always be the case, since an optimal policy might be a stochastic policy. This is especially true when there are states in the environment that are partially observable or indistinguishable from each other. Moreover, the aforementioned papers limit their focus to discrete state and action spaces.

In this Chapter, we study the addition of PBRS and PBA schemes to the reward, in

settings where: *i)* the optimal policy will be stochastic, and *ii)* state and action spaces may be continuous. We additionally provide guarantees on the convergence of an advantage actor-critic architecture that is augmented with a PBA scheme.

3.2 PBRS for Stochastic Policy Learning

The existing literature on PBRS has focused on augmenting value-based methods to learn optimal deterministic policies. In this section, we first show that PBRS preserves optimality, when the optimal policy is stochastic. Then, we show that the *learnability* will not be changed when using PBRS in soft Q-learning.

Proposition 1. *Assume that the optimal policy is stochastic. Then, with* $F := \gamma\phi(s_{t+1}) - \phi(s_t)$*, PBRS preserves the optimality of stochastic policies.*

Proof. The goal in the original MDP M was to find a policy π in order to maximize:

$$\pi^*_M = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\right]. \tag{3.1}$$

In PBRS, the goal is to determine a policy so that:

$$\begin{aligned}
\pi^*_{M'} &= \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{\infty}\gamma^t\big(R(s_t, a_t) + F(s_t, a_t, s_{t+1}, a_{t+1})\big)\big] \\
&= \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{\infty}\gamma^t\big(R(s_t, a_t) + \gamma\phi(s_{t+1}) - \phi(s_t)\big)\big] \\
&= \arg\max_\pi \left[\mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\big] - \mathbb{E}_{\tau\sim\pi}\big[\phi(s_0)\big]\right] \\
&= \arg\max_\pi \ \mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\big] - \int_s \rho_0(s)\phi(s)\mathrm{d}s.
\end{aligned} \tag{3.2}$$

The last term in Equation (3.2) is constant, and does not affect the identity of the maximizing policy of (3.1). □

Next, we examine the effect on learnability when using PBRS with soft Q-learning. Soft Q-learning is a value-based method for stochastic policy learning that was proposed in [30].

Different from Equation (3.1), the goal is to maximize both, the accumulated reward, and the policy entropy at each visited state:

$$\pi^*_{\text{soft}} = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{\infty} \gamma^t \big(R(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\big)\big]. \tag{3.3}$$

The entropy term $\mathcal{H}(\pi(\cdot|s_t))$ encourages exploration of the state space, and the parameter α is a trade-off between exploitation and exploration.

Before stating our result, we summarize the soft Q-learning update procedure. From [30], the optimal value-function, $V^*_{\text{soft}}(s_t)$, is given by:

$$V^*_{\text{soft}}(s_t) = \alpha \log \int_A \exp\big(\frac{1}{\alpha} Q^*_{\text{soft}}(s_t, a)\big)\mathrm{d}a. \tag{3.4}$$

The optimal soft Q-function is determined by solving a soft Bellman equation:

$$Q^*_{\text{soft}}\big(s_t, a_t\big) = r_t + \gamma\mathbb{E}_{s_{t+1}}\big[V^*_{\text{soft}}(s_{t+1})\big]. \tag{3.5}$$

The optimal policy can be obtained from Equation (3.5) as:

$$\pi^*_{\text{soft}}(a_t|s_t) = \exp\big(\frac{1}{\alpha}\big(Q^*_{\text{soft}}(s_t, a_t) - V^*_{\text{soft}}(s_t)\big)\big), \tag{3.6}$$

In the rest of this Section, we assume both, states and actions are discrete and no function approximator is used. We also omit subscripts for Q_{soft} and V_{soft}, and set $\alpha = 1$ for simplicity. From Equation (3.5), and as in Q-learning, soft Q-learning updates the soft Q-function by minimizing the soft Bellman error:

$$\delta Q_k(s_k, a_k) = r(s_k, a_k) + \gamma V_k(s_{k+1}) - Q_k(s_k, a_k), \tag{3.7}$$

where $V_k(s_{t+1}) = \log\sum_{a\in A}\exp\big(Q_k(s_{t+1}, a)\big)$. During training, $\pi_k(a_t|s_t) = \exp\big(Q_k(s_t, a_t) - V_k(s_t)\big)$. With λ denoting the learning rate, the Q-function update is given by:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \lambda\delta Q_k(s_k, a_k). \tag{3.8}$$

The main result of this section shows that the ability of an agent to learn an optimal policy is unaffected when using soft Q-learning augmented with PBRS. We define a notion of *learnability*, and use this to establish our claim.

During training, an agent encounters a sequence of states, actions, and rewards that serves as 'raw-data' which is fed to the RL algorithm. Let L and L' denote two RL agents. Let $\mathcal{D}_k = (s_k, a_k, r_k, s_{k+1})$ and $\mathcal{D}'_k = (s'_k, a'_k, r'_k, s'_{k+1})$ denote the experience tuple at learning step k from a trajectory used by L and L', respectively.

Definition 1 (Learnability)**.** *Denote the accumulated difference in the Q-functions of L and L' after learning for k steps by $\Delta Q_k(s,a)$ and $\Delta Q'_k(s,a)$, respectively. Then, given identical sample experiences, (that is, $\mathcal{D}_{k'} = \mathcal{D}'_{k'}\ \forall k' \leq k$), L and L' are said to have the same learnability if $\Delta Q_{k'}(s,a) = \Delta Q'_{k'}(s,a)\ \forall k' \leq k\ \forall s \forall a$.*

Proposition 2. *Soft Q-learning, with initial soft Q-values $Q(s,a) = Q_0(s,a)$ and augmented with PBRS where state potential is $\phi(s)$, has the same learnability as soft Q-learning without PBRS but with its soft Q-values initialized to $Q(s,a) = Q_0(s,a) + \phi(s)$.*

Proof. Consider an agent L that uses a PBRS scheme during learning and an agent L' that does not use PBRS, but has its soft Q-values initialized as $Q'_0(s,a) := Q_0(s,a) + \phi(s)$, where $Q_0(s,a)$ is the initial Q-value of L. We further assume that L and L' adopt the same learning rate. From Definition 1, to show that L and L' have the same learnability, we need to show that the soft Bellman errors $\delta Q_k(s_t, a_t)$ and $\delta Q'_k(s_k, a_k)$ are equal at each training step k, given the same experience sets $\mathcal{D}_k$ and $\mathcal{D}'_k$. From Equation (3.7), the soft Bellman errors for L and L' can be respectively written as:

$$\begin{aligned}\delta Q_k(s_k, a_k) = r(s_k, a_k) + \gamma\phi(s_{k+1}) - \phi(s_k) + \\ \gamma V_k(s_{k+1}) - Q_k(s_k, a_k)\end{aligned}$$
$$\delta Q'_k(s'_k, a'_k) = r(s'_k, a'_k) + \gamma V'_k(s'_{k+1}) - Q'_k(s'_k, a'_k).$$

Since $\mathcal{D}_{k'} = \mathcal{D}'_{k'}$ for each $k' \leq k$, comparing $\delta Q'_k(s_k, a_k)$ and $\delta Q_k(s'_k, a'_k)$ is reduced to comparing $\delta Q'_k(s_k, a_k)$ and $\delta Q_k(s_k, a_k)$. We show this by induction.

At training step $k = 0$ there is no update. Thus, $\delta Q_0(s_0, a_0) = \delta Q'_0(s_0, a_0)$. Assume that the Bellman errors are identical up to a step $k = K$. That is, $\delta Q_k(s_k, a_k) = \delta Q'_k(s_k, a_k)$ $\forall k \leq K$. Then, the accumulated errors for the two agents until this step are also identical.

That is, $\Delta Q_K(s,a) = \Delta Q'_K(s,a)\ \forall s \forall a$. Consider training step $k = K+1$. The state values at this step are: $V_K(s_{K+1}) = \log \sum_{a \in A} \exp\left[Q_0(s_{K+1},a) + \Delta Q_K(s_{K+1},a)\right]$ and $V'_K(s_{K+1}) = \log \sum_{a \in A} \exp\left[Q_0(s_{K+1},a) + \phi(s_{K+1}) + \Delta Q'_K(s_{K+1},a)\right]$ respectively. The Bellman errors at $k = K+1$ are:

$$\begin{aligned}\delta Q_{K+1}(s_K,a_K) &= r(s_K,a_K) + \gamma\phi(s_{K+1}) - \phi(s_K) \\ &\quad + \gamma V_K(s_{K+1}) - Q_K(s_K,a_K) \\ &= r(s_K,a_K) + \gamma\phi(s_{K+1}) - \phi(s_K) + \gamma V_K(s_{K+1}) \\ &\quad - Q_0(s_K,a_K) - \Delta Q_K(s_K,a_K)\end{aligned}$$

$$\begin{aligned}&\delta Q'_{K+1}(s_K,a_K) = r(s_K,a_K) + \gamma V'_K(s_{K+1}) - Q'_K(s_K,a_K) \\ &= r(s_K,a_K) + \gamma V'_K(s_{K+1}) - Q_0(s_K,a_K) - \phi(s_K) - \Delta Q'_K(s_K,a_K) \\ &= \delta Q_{K+1}(s_K,a_K) - \gamma\phi(s_{K+1}) + \gamma(V'_K(s_{K+1}) - V_K(s_{K+1})) \\ &= \delta Q_{K+1}(s_K,a_K) - \gamma\phi(s_{K+1}) + \gamma\phi(s_{K+1}) \\ &= \delta Q_{K+1}(s_K,a_K).\end{aligned}$$

It follows that $\Delta Q_{K+1}(s,a) = \Delta Q'_{K+1}(s,a)\ \forall s \forall a$. □

Remark 1. *If the Q-function is represented by a function approximator (as is typical for continuous action spaces), then Proposition 2 may not hold. This is because the Q-function in this scenario is updated using gradient descent, instead of Equation (3.8). Gradient descent is sensitive to initialization. Thus, different initial values may result in different updates of the Q-function.*

3.3 PBA for Stochastic Policy Learning

Although PBRS can preserve the optimality of policies, it suffers from the drawback of being unable to encode richer information, such as desired relations between states and actions. The authors of [94] proposed *potential-based advice* (PBA), a scheme that augments the potential function by including actions as an argument together with states. In this section,

we show that while using PBA, recovering the optimal policy can be difficult if the optimal policy is stochastic. Then, we propose a novel way to impart prior information in order to learn a stochastic policy with PBA.

3.3.1 Stochastic policy learning with PBA

Assume that we can compute $Q^*_M(s,a)$, the optimal value for state-action pair (s,a) in MDP M. The optimal stochastic policy for M is $\pi^*_M = \arg\max_{\tau\sim\pi} \mathbb{E}_\pi\big[Q^*_M(s,a)\big]$. From Equation (2.5), the optimal stochastic policy for the modified MDP M' that has its reward augmented with PBA is given by $\pi^*_{M'} = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\big[Q^*_M(s,a) - \phi(s,a)\big]$. Without loss of generality, $\pi^*_M \neq \pi^*_{M'}$. If the optimal policy is deterministic, then the policy for M can be recovered easily from that for M' using Equation (2.6). However, when it is stochastic, we need to average over trajectories in the MDP, which makes it difficult to recover the optimal policy for M from that of M'.

In the sequel, we will propose a novel way to take advantage of PBA in the policy gradient framework in order to directly learn a stochastic policy.

3.3.2 Imparting PBA in policy gradient

Let $J_M(\theta)$ denote the value of a parameterized policy π_θ in MDP M. That is, $J_M(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\right]$. Following the policy gradient theorem [78], and defining $G(s_t,a_t) := \sum_{i=t}^{i=\infty}\gamma^{i-t}r_i$, the gradient of $J(\theta)$ with respect to the parameter θ is given by:

$$\nabla_\theta J_M(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\big[G(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\big]. \tag{3.9}$$

Then, $\mathbb{E}_{\tau\sim\pi_\theta}\big[G(s_t,a_t)\big] = Q^{\pi_\theta}(s_t,a_t)$.

REINFORCE [78] is a policy gradient method that uses Monte Carlo simulation to learn θ, where the parameter update is performed only at the end of an episode (a trajectory of length T). If we apply a look-ahead PBA scheme as in Equation (2.3) along with REIN-

FORCE, then the total return from time t is given by:

$$\begin{aligned} G^a(s_t, a_t) &= \sum_{i=t}^{i=T} \gamma^{i-t} r_i + \gamma^{T-t}\phi(s_T, a_T) - \phi(s_t, a_t) \\ &= G(s_t, a_t) + \gamma^{T-t}\phi(s_T, a_T) - \phi(s_t, a_t). \end{aligned} \tag{3.10}$$

Notice that if $G^a(s_t, a_t)$ is used in Equation (3.9) instead of $G(s_t, a_t)$, then the policy gradient is biased. One way to resolve the problem is to add the difference $-\gamma^{T-t}\phi(s_T, a_T) + \phi(s_t, a_t)$ to $G^a(s_t, a_t)$. However, this makes the learning process identical to the original REINFORCE and PBA is not used. While using PBA in a policy gradient setup, it it important to add the term $\phi(s, a)$ so that the policy gradient is unbiased, and also leverage the advantage that PBA offers during learning.

To apply PBA in policy gradient, we turn to temporal difference (TD) methods. TD methods update estimates of the accumulated return based in part on other learned estimates, before the end of an episode. A popular TD-based policy gradient method is the actor-critic framework [78]. In this setup, after performing action a_t at step t, the accumulated return $G(s_t, a_t)$ is estimated by $Q_M(s_t, a_t)$ which, in turn, is estimated by $r_t + \gamma V_M(s_{t+1})$. It should be noted that the estimates are unbiased.

When the reward is augmented with look-ahead PBA, the accumulated return is changed to $Q_{M'}(s_t, a_t)$, which is estimated by $r_t + \gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t) + \gamma V_{M'}(s_{t+1})$. From Equation (2.5), at steady state, $Q_M(s_t, a_t) - Q_{M'}(s_t, a_t) = \phi(s_t, a_t)$. Intuitively, to keep policy gradient unbiased when augmented with look-ahead PBA, we can add $\phi(s_t, a_t)$ at each training step. In other words, we can use $r_t + \gamma\phi(s_{t+1}, a_{t+1}) + \gamma V_{M'}(s_{t+1})$ as the estimated return. It should be noted that before the policy reaches steady state, adding $\phi(s_t, a_t)$ at each time step will not cancel out the effect of PBA. This is unlike in REINFORCE, where the addition of this term negates the effect of using PBA. In the advantage actor-critic, an advantage term is used instead of the Q-function in order to reduce the variance of the estimated policy gradient. In this case also, the potential term $\phi(s_t, a_t)$ can be added in order to keep the policy gradient unbiased.

Algorithm 1 : Actor-critic augmented with PBA

Input: Differentiable policy function $\pi_\theta(a|s)$

Differentiable value function $V^\omega(s)$

Potential-based advice $\phi(s, a)$

Maximum episode T_{max}

Initialization:

policy parameter θ, value parameter ω, learning rate α^θ and α^ω, discount factor γ, episode counter $T \leftarrow 0$

repeat

initialize state s_0, $t \leftarrow 0$

repeat

Sample action $a_t \sim \pi_\theta(\cdot|s_t)$

Take action a_t, observe reward r_t, next state s_{t+1}

$$R = \begin{cases} 0, & \text{if } s_{t+1} \text{ is a terminal state ,} \\ V^\omega(s_{t+1}), & \text{otherwise.} \end{cases}$$

if use look-ahead advice **then**

$\delta_t = r_t + \gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t) + \gamma R - V^\omega(s_t)$

Update $\theta \leftarrow \theta + \alpha^\theta\big(\delta_t + \phi(s_t, a_t)\big)\nabla_\theta \log \pi_\theta(a_t|s_t)$

else

$\delta_t = r_t + \phi(s_t, a_t) - \gamma^{-1}\phi(s_{t-1}, a_{t-1}) + \gamma R - V^\omega(s_t)$

Update $\theta \leftarrow \theta + \alpha^\theta \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

end if

Update $\omega \leftarrow \omega - \alpha^\omega \delta_t \nabla_\omega V^\omega(s_t)$

until s_{t+1} is a terminal state

$T \leftarrow T + 1$

until $T > T_{max}$

A procedure for augmenting the advantage actor-critic with PBA is presented in Algorithm 1. α^θ and α^ω denote learning rates for the actor and critic respectively. When applying look-ahead PBA, at training step t, parameter ω of the critic $V^\omega(s)$ is updated as follows:

$$\begin{aligned}\delta_t^a &= r_t + \gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t) + \gamma V^\omega(s_{t+1}) - V^\omega(s_t)\\ \omega &= \omega - \alpha^\omega \delta_t^a \nabla_\omega V^\omega(s_t),\end{aligned}$$

where δ_t^a is the estimation error of the state value after receiving new reward $[r_t+\gamma\phi(s_{t+1}, a_{t+1})-\phi(s_t, a_t)]$ at step t. To ensure an unbiased estimate of the policy gradient, the potential term $\phi(s_t, a_t)$ is added while updating θ as:

$$\theta = \theta + \alpha^\theta \big(\delta_t^a + \phi(s_t, a_t)\big)\nabla_\theta \log \pi_\theta(a_t|s_t).$$

A similar method can be used when learning with look-back PBA. In this case, the critic and the policy parameter are updated as follows:

$$\begin{aligned}\delta_t^b &= r_t + \phi(s_t, a_t) - \gamma^{-1}\phi(s_{t-1}, a_{t-1}) + \gamma V^\omega(s_{t+1}) - V^\omega(s_t)\\ \omega &= \omega - \alpha^\omega \delta_t^b \nabla_\omega V^\omega(s_t),\\ \theta &= \theta + \alpha\big(\delta_t^b + \gamma^{-1}\mathbb{E}\big[\phi(s_{t-1}, a_{t-1})|s_t\big]\big)\nabla_\theta \log \pi_\theta(a_t|s_t)\end{aligned} \tag{3.11}$$

In fact, the potential term need not be added to ensure an unbiased estimate in this case. Then, the policy parameter update becomes:

$$\theta = \theta + \alpha\delta_t^b \nabla_\theta \log \pi_\theta(a_t|s_t), \tag{3.12}$$

which is exactly the policy update of the advantage actor-critic. This is formally stated in Proposition 3

Proposition 3. *When the actor-critic is augmented with look-back PBA, Equations (3.11) and (3.12) are equal in the sense of expectation. That is*

$$\begin{aligned}&\mathbb{E}_{(s_t,a_t)\sim\rho^{\pi_\theta}}\big[\big(\delta_t^b+\gamma^{-1}\mathbb{E}\big[\phi(s_{t-1}, a_{t-1})|s_t\big]\big)\nabla_\theta \log \pi_\theta(a_t|s_t)\big]\\ &\qquad = \quad \mathbb{E}_{(s_t,a_t)\sim\rho^{\pi_\theta}}\big[\delta_t^b \nabla_\theta \log \pi_\theta(a_t|s_t)\big],\end{aligned} \tag{3.13}$$

where ρ^{π_θ} is the distribution induced by the policy π_θ.

Proof. It is equivalent to show that:

$$\mathbb{E}_{(s_t,a_t)\sim\rho^{\pi_\theta}}\left[\mathbb{E}\big[\phi(s_{t-1},a_{t-1})|s_t\big]\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = 0. \tag{3.14}$$

The inner expectation $\mathbb{E}\big[\phi(s_{t-1},a_{t-1})|s_t\big]$ is a function of s_t, policy π_θ, and transition probability $\mathbb{T}$. Denoting this expectation by $f(s_t,\pi_\theta,\mathbb{T})$, we obtain:

$$\begin{aligned}
&\mathbb{E}_{(s_t,a_t)\sim\rho^{\pi_\theta}}\left[f(s_t,\pi_\theta,\mathbb{T})\nabla_\theta \log \pi_\theta(a_t|s_t)\right]\\
=&\mathbb{E}_{s_t\sim\rho^{\pi_\theta}}\left[\mathbb{E}_{a_t\sim\pi_\theta}\left[f(s_t,\pi_\theta,\mathbb{T})\nabla_\theta \log \pi_\theta(a_t|s_t)\right]\right]\\
=&\mathbb{E}_{s_t\sim\rho^{\pi_\theta}}\left[\int_A \pi_\theta(a_t|s_t) f(s_t,\pi_\theta,\mathbb{T})\frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}\mathrm{d}a\right]\\
=&\mathbb{E}_{s_t\sim\rho^{\pi_\theta}}\left[f(s_t,\pi_\theta,\mathbb{T})\nabla_\theta \int_A \pi_\theta(a_t|s_t)\mathrm{d}a\right] = 0.
\end{aligned} \tag{3.15}$$

The last equality follows from the fact that the integral evaluates to 1, and its gradient is 0. □

The main result of this chapter presents guarantees on the convergence of Algorithm 1 using the theory of 'two time-scale stochastic analysis' [7]. Assume that:

- **A1**: The value function $V^\omega(s)$ belongs to a linear family. That is, $V^\omega = \Phi\omega$, where $\Phi \in \mathbb{R}^{|S|\times k}, k < S$ is a known full-rank feature matrix, and $\omega \in \Omega \subseteq \mathbb{R}^k$.

- **A2**: For the set of policies $\{\pi_\theta, \theta \in \Theta \subseteq \mathbb{R}^d\}$, there exists a constant C_Θ such that $\|\nabla_\theta \log \pi_\theta\|_2 \leq C_\Theta$.

- **A3**: Learning rates of the actor and critic satisfy: $\sum_t \alpha_t^\theta = \sum_t \alpha_t^\omega = \infty$, $\sum_t[(\alpha_t^\theta)^2 + (\alpha_t^\omega)^2] < \infty$, $\lim_{t\to\infty} \frac{\alpha_t^\theta}{\alpha_t^\omega} = 0$.

Definition 2. *For a probability measure μ on a finite set $\mathcal{M}$, the ℓ_2-norm of a function f with respect to μ is defined as* $||f||_\mu := \left[\int_{\mathcal{M}} |f(X)|^2 d\mu(X)\right]^{\frac{1}{2}} = \left[\mathbb{E}_\mu(|f(X)|^2)\right]^{\frac{1}{2}}$.

Theorem 1 gives a bound on the error introduced as a result of approximating the value function $V_{M'}$ with $V_{M'}^{\omega}$ as in assumption **A1**. This error term is small if the family Ω is rich. In fact, if the critic is updated in batches, a tighter bound can be achieved, as shown in Proposition 1 of [103]. Extending the result to the case of online updates is a subject of future work.

Theorem 1. *Let* $\mathcal{E}(\theta) := \left\| V_{M'}^{\omega(\theta)}(s) - V_{M'}^{\pi_\theta}(s) \right\|_{\rho^{\pi_\theta}}$. *Then, for any limit point* $(\theta^*, \omega^*) := \lim_{T_{max}\to\infty} (\theta_{T_{max}}, \omega_{T_{max}})$ *of Algorithm 1,* $\|\nabla_\theta J_M(\theta^*)\|_2 \leq C\mathcal{E}(\theta^*)$.

Proof. We consider only look-ahead PBA. The proof for look-back PBA follows similarly. Define $F := F(s, a, s', a')$. From assumption **A3**, the actor is updated at a slower rate than the critic. This allows us to fix the actor to study the asymptotic behavior of the critic [4]. The update dynamics of the critic can be represented by:

$$\dot{\omega} = \mathbb{E}_{\rho^{\pi_\theta}}\left[\delta_\omega \nabla_\omega V_{M'}^{\omega}(s)\right], \tag{3.16}$$

where $\delta_\omega = r(s,a) + \gamma\phi(s',a') - \phi(s,a) + \gamma V^{\omega}(s') - V^{\omega}(s)$ if look-ahead PBA is applied. When the critic is approximated by a linear function (assumption **A1**), ω will converge to $\omega(\theta)$, an asymptotically stable equilibrium of Equation (3.16). The update of the actor is then:

$$\dot{\theta} = \mathbb{E}_{\rho^{\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(a|s)\big(r(s,a) + F + \gamma V_{M'}^{\omega(\theta)}(s') + \phi(s,a)\big)\right]. \tag{3.17}$$

Let Θ_s denote the set of asymptotic stable equilibria in Equation (3.17). Any $\theta \in \Theta_s$ will satisfy $\dot{\theta} = 0$ in Equation (3.17). Then, $\{(\theta_t, \omega_t)\}_{t>0}$ will converge to $\{(\theta, \omega(\theta)) : \theta \in \Theta_s\}$.

Now, consider the evaluation of π_θ, $\theta \in \Theta_s$, in the original MDP M. We obtain the following equations:

$$\begin{aligned}
\nabla_\theta J_M(\theta) &= \mathbb{E}_{\rho^{\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(a|s) Q_M^{\pi_\theta}(s,a)\right] \\
&= \mathbb{E}_{\rho^{\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(a|s)\big(Q_{M'}^{\pi_\theta}(s,a) + \phi(s,a)\big)\right] \\
&= \mathbb{E}_{\rho^{\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(a|s)\big(r(s,a) + F + \gamma V_{M'}^{\pi_\theta}(s') + \phi(s,a)\big)\right].
\end{aligned} \tag{3.18}$$

Subtracting Equation (3.17) from Equation (3.18), and applying the Cauchy-Schwarz inequality to the result yields:

$$\nabla_\theta J_M(\theta) = \gamma \mathbb{E}_{\rho^{\pi_\theta}} \left[\nabla_\theta \log \pi_\theta(a|s) \big(V_{M'}^{\omega(\theta)}(s') - V_{M'}^{\pi_\theta}(s') \big) \right]$$
$$\therefore \left\| \nabla_\theta J_M(\theta) \right\|_2 \leq \gamma \left\| \nabla_\theta \log \pi_\theta(a|s) \right\|_{\rho^{\pi_\theta}} \left\| V_{M'}^{\omega(\theta)}(s) - V_{M'}^{\pi_\theta}(s) \right\|_{\rho^{\pi_\theta}}.$$

The result follows by applying assumption **A2**. □

Remark 2. *Look-back PBA could result in better performance compared to look-ahead PBA since look-back PBA does not involve estimating a future action.*

3.4 Experiments

Our experiments seek to compare the performance of an actor-critic architecture augmented with PBA and with PBRS with the 'vanilla' advantage actor-critic (A2C). We consider two setups. The first is a *Puddle-Jump Gridworld* [59], where the state and action spaces are discrete. The second environment we study is a continuous state and action space *mountain car* [8].

In each experiment, we compare the rewards received by the agent when it uses the following schemes: *i)*: 'vanilla' (A2C); *ii)*: A2C augmented with PBRS; *iii)*: A2C with look-ahead PBA; *iv)*: A2C with look-back PBA.

3.4.1 Puddle-Jump Gridworld

Figure 3.1 depicts the *Puddle-jump gridworld* environment as a 10x10 grid. The state space is $s = (x, y)$ denoting the position of the agent in the grid, where $x, y \in \{0, 1, \ldots, 9\}$. The goal of the agent is to navigate from the start state $S = (0, 0)$ to the goal $G = (9, 9)$. At each step, the agent can choose from actions in the set $A = \{up, down, left, right, jump\}$. There is a *puddle* along row 2 which the agent should jump over. Further, the states $(9, 8)$ and $(8, 9)$ (blue squares in Figure 3.1) are indistinguishable to the agent. As a result, any optimal policy for the agent is a stochastic policy. If the *jump* action is chosen in rows 3

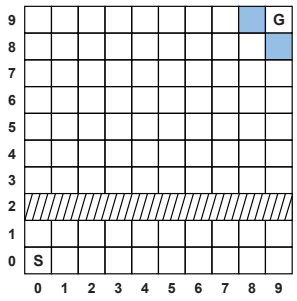


Figure 3.1: Schematic of the puddle-jump gridworld. The state of the agent is its position (x, y). The shaded row (row 2) represents the puddle the agent should jump over. The two blue grids denote states that are indistinguishable to the agent. The agent can choose an action from the set $\{up, down, left, right, jump\}$ at each step.

or 1, the agent will land on the other side of the puddle with probability p_j, and remain in the same state otherwise. This action chosen in other rows will keep the agent in its current state. Any action that will move the agent off the grid will keep its state unchanged. The agent receives a reward of -0.05 for each action, and $+1000$ for reaching G.

When using PBRS, we set $\phi^{PBRS}(s) := u_0$ for states in rows 0 and 1, and $\phi^{PBRS}(s) := u_1$ for all other states. We need $u_1 > u_0$ to encourage the agent to jump over the puddle. Unlike in PBRS, PBA can provide the agent with more information about the actions it can take. We set $\phi^{PBA}(s, a)$ to a 'large' value if action a at state s results in the agent moving closer to the goal according to the ℓ_1 norm, $\big(|G - x| + |G - y|\big)$. We additionally stipulate that $\frac{1}{|A|}\sum_{a\in A}\phi^{PBA}(s, a) = \phi^{PBRS}(s)$. That is, the state potential of PBA is the same as the state potential of PBRS under a uniform distribution over the actions. This is to ensure a fair comparison between PBRS and PBA.

In our experiment, we set the discount factor $\gamma = 1$. Since the dimensions of the state and action spaces is not large, we do not use a function approximator for the policy π. A parameter $\theta_{s,a}$ is associated to each state-action pair, and the policy is computed as: $\pi_\theta(a|s) = \frac{\exp(\theta_{s,a})}{\sum_{a\in A}\exp(\theta_{s,a})}$. We fix $\alpha^\omega = 0.001$, and $\alpha^\theta = 0.2$ for all cases. From Figure 3.2,

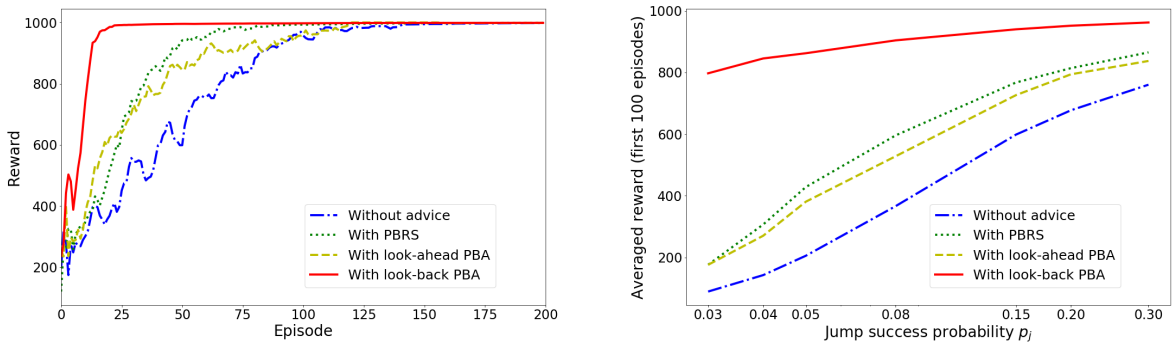


Figure 3.2: The figure on the left shows average rewards in puddle-jump gridworld when jump success probability $p_j = 0.2$. The baseline is the advantage actor-critic without advice. The figure on the right shows the average reward for the first 100 episodes with respect to the jump success probability p_j.

we observe that the look-back PBA scheme performs the best, in that the agent converges to the goal in **five times** fewer episodes (25 vs. 125 episodes) than A2C without advice. When A2C is augmented with PBRS, convergence to the goal is slightly faster than without any reward shaping. When augmented with look-ahead PBA, in the first few episodes, the reward increases faster than in the case of A2C augmented with PBRS. However, this slows down after the early training stages and the policy converges to the goal in about the same number of episodes as a policy trained without advice. A reason for this could be that during later stages of training, a look-ahead PBA scheme might advise an agent with 'bad' actions, leading to bad policies, thereby impeding the progress of learning. For example, an action a_t might be a good choice at state s_t, but the look-ahead PBA scheme might indicate that a_t is bad, due to a poor estimate of the future action a_{t+1}.

A smaller jump success probability p_j is an indication that it is more difficult for the agent to reach the goal state G. Figure 3.2 shows that look-back PBA results in the highest reward for a more difficult task (lower p_j), when compared with the other reward shaping schemes.

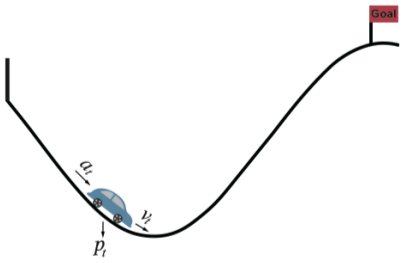


Figure 3.3: Schematic of the mountain-car environment. The agent's state is represented by its position p_t (along the $x-$coordinate) and velocity v_t. The action a_t is a force applied to the car. The goal is marked as a flag.

3.4.2 Continuous Mountain Car

In the mountain car (MC) environment, an under powered car in a valley has to drive up a steep hill to reach the goal. In order to achieve this, the car should learn how to accumulate momentum. A schematic for this environment is shown in Figure 3.3. This MC environment has continuous state and action spaces. The state $s = (p, v)$ denotes position $p \in [-1.2, 0.6]$ and velocity $v \in [-0.07, 0.07]$. The action $a \in [-1, +1]$. The continuous action space makes it difficult to use classic value-based methods, such as Q-learning and Sarsa-learning. The reward provided by the environment depends on the action and whether the car reaches the goal. Specifically, once the car reaches the goal it receives $+100$, and before that, the reward at time t is $-|a_t|^2$. This reward structure therefore discourages the waste of energy. This acts as a barrier for learning, because there appears to be a sub-optimal solution where the agent remains at the bottom of the valley. Moreover, the reward for reaching the goal is significantly delayed, which makes it difficult for the conventional actor-critic algorithm to learn a good policy.

One choice of a potential function while using PBRS in this environment is $\phi^{PBRS}(s_t) := p_t + 2$, where the offset is so that the potential is positive. An interpretation of this scheme is: *'state value is larger when the car is horizontally closer to the goal.'* The PBA scheme we use

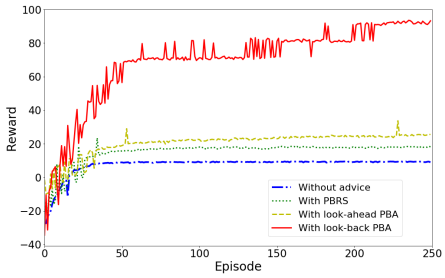


Figure 3.4: Average rewards for continuous mountain car problem (averaged over 10 different environment random seeds). The baseline is the A2C without advice.

for this environment encourages the accumulation of momentum by the car– the direction of the action is encouraged to be the same as the current direction of the car's velocity. In the meanwhile, we discourage inaction. Mathematically, the potential advice function has a larger value if $a_t \neq 0$. We let $\phi^{PBA}(s_t, a_t) = 1$, if $a_t v_t > 0$, and $\phi^{PBA}(s_t, a_t) = 0$, otherwise.

In our experiments, we set $\gamma = 0.99$. To deal with the continuous state space, we use a neural network (NN) as a function approximator. The policy distribution $\pi_\theta(a|s)$ is approximated by a normal distribution, the mean and variance of which are the outputs of the NN. The value function is also represented by an NN. We set $\alpha^\theta = 1 \times 10^{-5}$ and $\alpha^\omega = 5.6 \times 10^{-4}$, and use Adam [40] to update the NN parameters. The results we report are averaged over 10 different environment seeds.

No advice	PBRS	Look-ahead PBA	Look-back PBA
10%	20%	40%	100%

Table 3.1: Percentage of trials where policy converges correctly in continuous mountain car.

Our experiments indicate that the policy makes the agent converge to one of two points: the goal, or remain stationary at the bottom of the valley. The percentage of solutions that

converge to the goal is shown in Table 3.1. From Figure 3.4 and Table 3.1, when learning with the vanilla A2C, the agent is able to reach the goal only in 10% of the trials (out of 10 trials), and was stuck at the sub-optimal solution for the remaining trials. With PBRS, the agent could converge correctly in only 20% of the trials. This is because the agent might have to take an action that moves it away from the goal in order to accumulate momentum. However, the potential function $\phi^{PBRS}(\cdot)$ discourages such actions. In comparison, the average reward when using look-ahead PBA is slightly higher, and similar to the gridworld setup, look-back PBA performs the best, where the agent is able to reach the goal in 100% of the trials.

3.5 Summary

This chapter presented a framework for augmenting the reward received by an RL agent with PBRS and with PBA. Different from prior work, we demonstrated that our approach can be used in environments with continuous states and actions, and when the optimal policy is stochastic. We presented guarantees on the convergence of an algorithm that augments an A2C architecture with these schemes. Our experiments indicated that these schemes allowed the agent to achieve higher average rewards, and learn an optimal policy faster.

Chapter 4

POTENTIAL-BASED ADVICE IN MULTI-AGENT REINFORCEMENT LEARNING

4.1 Motivation

Multi-agent reinforcement learning (MARL) involves multiple autonomous agents, all of whom share a common environment [10]. Examples of multi-agent systems where MARL has been applied include autonomous vehicle coordination [73], multi-player video games [80], and analysis of social dilemmas [46].

In these settings, any single agent interacts not only with the environment, but also with other agents. Since actions of an agent will affect the behavior of others, the environment becomes non-stationary from the perspective of any single agent. Thus, independently learning individual agents, who assume other agents to be part of the environment, can result in unstable learning regimes [23, 60, 81].

When dimensions of the state and action spaces are large, or when agents are unable to communicate with each other, agents will need to learn decentralized policies using only their individual observations. Decentralized policies can be learned in a centralized manner by adopting the *centralized training with decentralized execution (CTDE)* paradigm [24, 55]. During training, an agent can make use of information about other agents' states and actions to aid its own learning. At test-time, decentralized policies learned during training are executed.

Another challenge when reward signals are sparse is that of *credit assignment*- an agent may not receive immediate feedback on the quality of an action that it takes in some state. Moreover, when there are multiple agents, each agent will require an indicator of how its own action contributed to getting a reward *vis-a-vis* other agents [12]. In cooperative tasks,

difference rewards [18, 86, 97] have been effective in assigning credit to agents that contribute positively towards accomplishing the objective, and punishing others. However, credit assignment in more general mixed cooperative-competitive deep MARL environments remains an open problem.

In this chapter, we introduce *Shaping Advice in Multi-agent deep reinforcement learning (SAM)* to address credit assignment in diverse deep MARL environments. SAM incorporates information about the environment and task into an actor-critic framework and has the following properties.

First, SAM uses an individualized centralized critic for each agent to learn from that agent's experiences. This critic is required only during the training phase, and uses information about states and actions of all agents. The critic aids in learning decentralized agent policies, which solely use local observation-action information.

Next, SAM exploits information about the task and environment to define *shaping advice* for each agent. This advice is added to the reward received by the agent from the environment at each time step. The advice is potential-based [15, 94], and provides the agent with feedback on the quality of its actions in a given state.

The centralized critic will facilitate training with shaping advice derived from states and actions of all agents. SAM learns decentralized policies in environments with continuous state and action spaces. The advice in SAM can be interpreted as *domain knowledge* that aids credit assignment [57]. This advice only needs to be specified once at the start of the training process. Further, it can easily be specified by non-experts, which distinguishes SAM from other approaches to credit assignment like learning from demonstrations [35] and imitation learning [38, 71].

Finally, SAM can be applied to cooperative and competitive multi-agent frameworks. Shaping advice provided by SAM does not distract agents from accomplishing task objectives specified by environment rewards. We demonstrate theoretically that convergence of policy gradients and values when using SAM implies convergence of these quantities for the task specified only by the environment reward.

We evaluate SAM on the *Cooperative Navigation, Physical Deception, and Predator-Prey* tasks in the particle world environment [55]. These tasks have sparse rewards, and include cooperative and competitive cases. Our results show that SAM accelerates learning of policies, and results in improved agent performance. In competitive tasks, SAM alleviates a problem of 'oscillating rewards' reported in [55]. We observe that policies of agents equipped with SAM adapt better to each other than in the absence of SAM, resulting in more stable rewards.

4.2 Shaping Advice in Multi-Agent Reinforcement Learning

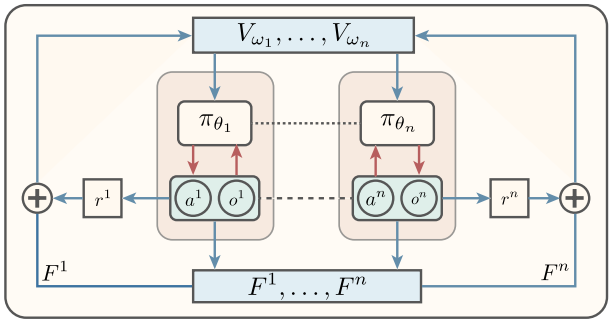


Figure 4.1: Schematic of SAM. A centralized critic estimates value functions $V_{\omega_1}, \ldots, V_{\omega_n}$. Actions for an agent i are sampled from its policy π_{θ_i} in a decentralized manner. Actions of all agents along with the state are used to determine *shaping advice* $F^1, \ldots, F^n$. The advice F^i is augmented to the environment reward r^i. The workflow shown by blue arrows in the outer box is required only during training. During execution, only the workflow shown by the red arrows inside the inner boxes is needed.

Our goal is to devise a method under the following conditions: *i)* policies learned by agents can only use local observations; *ii)* sharing of parameters among agents during training is allowed; *iii)* a model of the environment is not available; *iv)* agents cannot communicate with each other; *v)* we allow the use of a mechanism during training that provides agents

with immediate feedback on their actions.

We introduce ***SAM*** to address credit assignment in cooperative and competitive MARL environments. We augment *shaping advice* to the reward supplied by the environment for each agent to provide immediate feedback on its actions. Figure 4.1 shows a schematic of SAM.

4.2.1 Centralized Critic

SAM uses a centralized critic during the training phase. Information about states (or observations) and actions of all agents is used to learn a decentralized policy for each agent, that uses only local agent observations. One way to do this is by using an actor-critic framework, which combines policy gradients with *temporal difference (TD)* techniques.

At time t, the joint action $(a_t^1, \ldots, a_t^n)$ is used to estimate the accumulated return for each agent i as $r_t^i + \gamma V^i(s_{t+1})$. This quantity is called the *TD-target*. Subtracting $V^i(s_t)$ from the TD-target gives the *TD-error*, which is an unbiased estimate of the advantage. Each actor can then follow a gradient based on this TD-error.

Although we learn a separate critic for each agent like in [55], the credit-assignment problem remains to be addressed. In cooperative MARL, difference rewards [24, 97] have been effective in ensuring credit assignment, but this requires a user-chosen default action. This can be difficult to choose in continuous action spaces. To alleviate this problem, SAM uses a *potential-based* shaping heuristic to *advise* agents, in order to perform credit assignment in cooperative and competitive deep MARL environments.

4.2.2 Shaping Advice in Multi-Agent Actor-Critic

We describe how to augment shaping advice to the multi-agent policy gradient to assign credit. We use the actor-critic framework with a centralized critic and decentralized actors.

For an agent i, shaping advice F^i is augmented to the environment reward r^i at each time step. F^i is specified by a difference of potentials (Eqn. (2.7) or (2.8)). The centralized critic allows using states and actions of all agents to specify F^i. Using look-ahead advice,

Q-values in the modified game $\mathcal{G}'$ with reward $R^i + F^i$ and original game $\mathcal{G}$ with reward R^i are related as [94]:

$$[Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})]_{\mathcal{G}} = [Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})]_{\mathcal{G}'} + \phi_i(s_t, a_t^i, a_t^{-i}) \tag{4.1}$$

The accumulated return for agent i is then estimated by $r_t^i + \gamma\phi_i(s_{t+1}, a_{t+1}^i, a_{t+1}^{-i}) - \phi_i(s_t, a_t^i, a_t^{-i}) + \gamma V^i(s_{t+1})$. From Equation (4.1), we can add $\phi_i(s_t, a_t^i, a_t^{-i})$ to the TD-target at each time step to keep the policy gradient unbiased. A detailed description of the SAM algorithm is provided in algorithm 2.

Assume that the critic and actor in SAM for agent i are respectively parameterized by ω_i and θ_i. When the actor is updated at a slower rate than critic, asymptotic behavior of the critic can be analyzed by keeping the actor fixed using *two time-scale stochastic approximation* methods [6]. For agent i, the TD-error at time t is:

$$\delta_t^i := r_t^i + F_t^i + \gamma V_{\omega_i}(s_{t+1}) - V_{\omega_i}(s_t). \tag{4.2}$$

The update of the critic can be expressed as a first-order ordinary differential equation (ODE) in ω_i, given by:

$$\dot{\omega}_i = \mathbb{E}_{\boldsymbol{\pi_\theta}}[\delta^i \nabla_{\omega_i} V_{\omega_i}(s_t)] \tag{4.3}$$

Under an appropriate parameterization of the value function, this ODE will converge to an asymptotically stable equilibrium, denoted $\omega_i(\boldsymbol{\theta})$. At this equilibrium, the TD-error for agent i is $\delta^i_{t,\omega_i(\boldsymbol{\theta})} = r_t^i + F_t^i + \gamma V_{\omega_i(\boldsymbol{\theta})}(s_{t+1}) - V_{\omega_i(\boldsymbol{\theta})}(s_t)$.

The update of the actor can then be determined by solving a first order ODE in θ_i. With look-ahead advice, a term corresponding to the shaping advice at time t will have to be added to ensure an unbiased policy gradient (Equation (4.1)). This ODE can be written as:

$$\dot{\theta}_i = \mathbb{E}_{\boldsymbol{\pi_\theta}}[(\delta^i_{t,\omega_i(\boldsymbol{\theta})} + \phi_i(s_t, a_t^i, a_t^{-i}))\nabla_{\theta_i} \log\ \pi_{\theta_i}(a_t^i|o_t^i)] \tag{4.4}$$

A detailed analysis of the convergence of the actor and critic parameters is presented in [101]. This two-timescale procedure demonstrates convergence of parameters in the stochastic game with the shaped reward, $\mathcal{G}'$.

Algorithm 2 SAM: Shaping Advice in Multi-agent deep reinforcement learning

Input: For each agent i: parameters θ_i (for agent policy), ω_i (for agent value function); Shaping advice $\phi_i(s, a^i, a^{-i})$. Learning rates $\alpha^\theta, \alpha^\omega$; Episode limit T_{max}. Initialize $T = 0$

repeat

$t \leftarrow -1$; $\phi_i(s_{-1}, a^i_{-1}, a^{-i}_{-1}) = 0$ for all i

Initialize state information s_0

repeat

$t \leftarrow t + 1$

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

sample $a^i_t \sim \pi_{\theta_i}(\cdot|o^i_t)$

end for

Take action $a_t = [a^1_t, \ldots, a^n_t]$, observe new state information s_{t+1} and obtain reward r^i_t for each agent. Use a_t to determine $\phi_i(s_t, a_t)$ for all agents

if s_{t+1} is terminal **then**

$V_{\omega_i}(s_{t+1}) = 0$

end if

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

if look-ahead advice **then**

estimate $a_{t+1} = [a^1_{t+1}, \ldots, a^n_{t+1}]$

$F^i_t := \gamma\phi_i(s_{t+1}, a^i_{t+1}, a^{-i}_{t+1}) - \phi_i(s_t, a^i_t, a^{-i}_t)$

TD-error: $\delta^i_t := r^i_t + F^i_t + \gamma V_{\omega_i}(s_{t+1}) - V_{\omega_i}(s_t)$

$\tilde{\delta}^i_t := \delta^i_t + \phi_i(s_t, a^i_t, a^{-i}_t)$

else

$F^i_t := \phi_i(s_t, a^i_t, a^{-i}_t) - \gamma^{-1}\phi_i(s_{t-1}, a^i_{t-1}, a^{-i}_{t-1})$

TD-error: $\delta^i_t := r^i_t + F^i_t + \gamma V_{\omega_i}(s_{t+1}) - V_{\omega_i}(s_t)$

$\tilde{\delta}^i_t := \delta^i_t$

end if

Update actor: $\theta_i \leftarrow \Gamma_i[\theta_i + \alpha^\theta_t \tilde{\delta}^i_t \nabla_{\theta_i} \log\ \pi_{\theta_i}(a^i_t|o^i_t)]$

Update critic: $\omega_i \leftarrow \omega_i - \alpha^\omega_t \delta^i_t \nabla_{\omega_i} V_{\omega_i}(s_t)$

end for

until s_{t+1} is a terminal state

$T \leftarrow T + 1$

until $T > T_{max}$

However, our goal is to provide a guarantee of convergence in the original game $\mathcal{G}$. We establish such a guarantee assuming that parameterization of the value function results in small errors, and policy gradients are bounded.

Theorem 2. *In the stochastic game $\mathcal{G}'$, let $||V_i^{\pi_{\boldsymbol{\theta}}}(s) - V_{\omega_i(\boldsymbol{\theta})}(s)||_{\pi_{\boldsymbol{\theta}}} \leq \mathcal{E}_i(\boldsymbol{\theta})$, and let $||\nabla_{\theta_i} \log \pi_{\theta_i}||_{\pi_{\boldsymbol{\theta}}} \leq C_i(\boldsymbol{\theta})$. Let $(\boldsymbol{\theta}^*, \omega(\boldsymbol{\theta})^*)$ be the set of limit points of SAM. Then, in the original stochastic game $\mathcal{G}$, for each agent i, $||\nabla_{\theta_i} J_i(\boldsymbol{\theta}^*)||_2 \leq C_i(\boldsymbol{\theta}^*)\mathcal{E}_i(\boldsymbol{\theta}^*)$.*

Proof: Let $\Theta_{i_{eq}}$ denote the set of asymptotically stable equilibria of the ODE in θ_i. Let $\Theta_{eq} := \Theta_{1_{eq}} \times \cdots \times \Theta_{n_{eq}}$. Then, in the set Θ_{eq}, $\dot{\theta}_i = 0$ for each agent i.

Consider a policy $\pi_{\boldsymbol{\theta}}$, $\boldsymbol{\theta} \in \Theta_{eq}$. In the original game $\mathcal{G}$,

$$\nabla_{\theta_i} J_i(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\pi_\theta}}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_t^i|o_t^i) Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})] \tag{4.5}$$

From Equation (4.1), $[Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})]_{\mathcal{G}} = [Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})]_{\mathcal{G}'} + \phi_i(s_t, a_t^i, a_t^{-i})$. Since we use an advantage actor critic, we replace $[Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})]_{\mathcal{G}'}$ with an advantage term, defined as $[Q_i^{\boldsymbol{\pi_\theta}}(s_t, a_t^i, a_t^{-i})]_{\mathcal{G}'} - V_i^{\boldsymbol{\pi_\theta}}(s_t)$. Substituting these quantities in Equation (4.5),

$$\begin{aligned} \nabla_{\theta_i} J_i(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\pi_\theta}}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_t^i|o_t^i). \\ (r_t^i + F_t^i + \gamma V_i^{\pi_{\boldsymbol{\theta}}}(s_{t+1}) \\ - V_i^{\boldsymbol{\pi_\theta}}(s_t) + \phi_i(s_t, a_t^i, a_t^{-i}))] \end{aligned} \tag{4.6}$$

At equilibrium, $\dot{\theta}_i = 0$ in Equation (4.4). Subtracting this from Equation (4.6),

$$\begin{aligned} \nabla_{\theta_i} J_i(\boldsymbol{\theta}) - \dot{\theta}_i = \nabla_{\theta_i} J_i(\boldsymbol{\theta}) \\ = \mathbb{E}_{\boldsymbol{\pi_\theta}}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_t^i|o_t^i). \\ (\gamma(V_i^{\pi_{\boldsymbol{\theta}}}(s_{t+1}) - V_{\omega_i(\boldsymbol{\theta})}(s_{t+1})) \\ - (V_i^{\pi_{\boldsymbol{\theta}}}(s_t) - V_{\omega_i(\boldsymbol{\theta})}(s_t)))] \end{aligned}$$

Using the Cauchy-Schwarz inequality,

$$||\nabla_{\theta_i} J_i(\boldsymbol{\theta}^*)||_2 \leq |\gamma - 1|.||V_i^{\pi_{\boldsymbol{\theta}}}(s) - V_{\omega_i(\boldsymbol{\theta})}(s)||_{\pi_{\boldsymbol{\theta}}}.$$
$$||\nabla_{\theta_i} \log \pi_{\theta_i}||_{\pi_{\boldsymbol{\theta}}}$$
$$\leq C_i(\boldsymbol{\theta}^*)\mathcal{E}_i(\boldsymbol{\theta}^*) \tag{4.7}$$

Each term on the right side of Eqn. (4.7) is bounded. Thus, $J_i(\boldsymbol{\theta})$ converges for each agent i in the original game $\mathcal{G}$, even though policies are synthesized in the modified game $\mathcal{G}'$. □

Proposition 2 demonstrates that the additional reward provided by SAM to guide the agents does not distract them from accomplishing the task objective that is originally specified by the environment reward.

4.3 Experiments

4.3.1 Enviroment and Task Descriptions

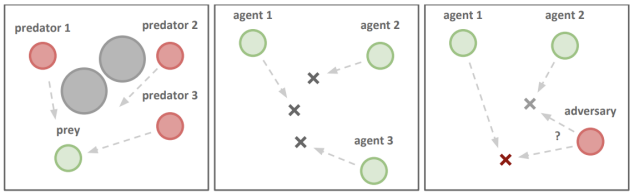


Figure 4.2: Representations of tasks from the Particle World Environment [55] that we study. (*Left to Right*) Predator-Prey (PP), Cooperative Navigation (CN), and Physical Deception (PD). In PP, predators (red) seek to catch the prey (green) while avoiding obstacles (grey). The prey can move faster than predators. In CN, agents (green) each seek to navigate to a different landmark (×) and are penalized for collisions with each other. In PD, one of the agents (green) must reach the true landmark (red ×), while preventing the adversary from reaching this landmark. In all tasks, rewards are *sparse*. Agents receive a reward or penalty only when a corresponding reachability or collision criterion is satisfied. Agents do not obtain a reward at other time steps.

We examine three tasks from the *Particle World* environment [55] where multiple agents share a two-dimensional space with continuous states and actions, and discrete time. An illustration of the tasks is shown in Figure 4.2. These include cooperative and competitive objectives. In each task, agents only receive a reward or penalty when encountering other agents or landmarks. No reward is provided at other time steps. Although the sparse reward structure precisely defines the task objective, this also makes it difficult for agents to receive immediate feedback on the quality of their actions at each step.

Predator-Prey

This task has N predator agents trying to capture 1 prey [55]. The prey can move faster than the predators, but predators can coordinate their behaviors. Predators are rewarded when one of them collides with the prey, while the prey is penalized for the same. The prey is also penalized if it exits from corners of the environment. Reward at other times is zero. L landmarks impede movement of the agents. Each agent observes relative positions and velocities of other agents, and positions of landmarks.

Cooperative Navigation

This task has N agents and N landmarks [55]. Positions of landmarks and initial positions of the agents are randomly generated. Landmarks are assumed to be immovable. Each agent observes the relative positions of other agents and landmarks. Agents are each rewarded r when an agent reaches a landmark, and penalized for collisions with each other. At all other times, the reward is zero. Therefore, the maximum rewards that the agents can obtain in this task is rN. Thus, agents must learn to *cover* all the landmarks, and not collide with each other.

Physical Deception

This task has 1 adversary, N agents, and N landmarks [55]. All agents observe positions of landmarks and other agents. Only one landmark is the true target. Positions of landmarks and initial positions of all agents are randomly generated. Locations of landmarks are assumed to be fixed in an episode. Good agents are rewarded when any one reaches the target landmark, and penalized if the adversary reaches the target. At all other times, the agents get a reward of zero. An adversary also wants to reach the target, but it does not know which landmark is the target landmark. Thus, good agents have to learn to split up and cover the landmarks in order to deceive the adversary.

4.3.2 Advising Agents with SAM

In each environment, SAM provides shaping advice to guide an agent in the direction of higher positive reward. This advice is augmented to the reward received from the environment. Since we use a centralized critic, actions taken by other agents are also available to any one agent during training. The advice is a heuristic given by a difference of potential functions (Equations (2.7) or (2.8)), and only needs to be specified once at the start of the training process.

During training, at each time step, an agent has knowledge of positions of other agents and landmarks, relative to itself, and actions taken by other agents. The objective of the task and geometrical features of the environment are used to design *shaping advice* for the agent. We summarize the properties that this advice must satisfy in each case.

In *Predator-Prey*, shaping advice for predators will be positive when they take actions that move them towards the prey, and negative if they move away. In *Cooperative Navigation*, the shaping advice will be positive when the agents take actions that move them towards different landmarks. In *Physical Deception*, the adversary's observation does not indicate which landmark is the true one. Thus, agents spreading out can deceive the adversary. Advice for agents in this task is similar to that in Cooperative Navigation.

Task	$\phi_i(s_t, a_t^i, a_t^{-i})$**: SAM-Uniform**	$\phi_i(s_t, a_t^i, a_t^{-i})$**: SAM-NonUniform**
CN	$\alpha_1 exp(-\beta_1 \sum_{j=1}^{N} dist(s_t^j, L_j))$	$-M_1 \theta_{a_t^i L_i} + \alpha_2 exp(-\beta_2 \sum_{j=1}^{N} dist(s_t^j, L_j))$
PD	$\alpha_3 exp(-\beta_2 \sum_{j=1}^{N} dist(s_t^j, L_j))$	$-M_2 \theta_{a_t^i L_i} + \alpha_4 exp(-\beta_4 \sum_{j=1}^{N} dist(s_t^j, L_j))$
PP	$\alpha_5 exp(-\beta_5 \sum_{j=1}^{N} dist(s_t^{pred_j}, s_t^{prey}))$	$-M_3 \sum_{j=1}^{N} \theta_{a_t^{pred_j} s_t^{prey}} + \alpha_6 exp(-\beta_6 \sum_{j=1}^{N} dist(s_t^{pred_j}, s_t^{prey}))$

Table 4.1: Shaping advice, F_t^i provided by SAM is given by Equation (2.7) or (2.8). The table lists the form of the potential functions used in the Cooperative Navigation (CN), Physical Deception (PD), and Predator-Prey (PP) tasks. L_j is the landmark to which agent j is *anchored* to. $dist(\cdot, \cdot)$ denotes the Euclidean distance. $\theta_{a_t^j L_j} \in [0, \pi]$ is the angle between the direction of the action taken by agent j and the vector directed from its current position to L_j. In *SAM-Uniform*, advice for every action of the agents in a particular state is the same. In *SAM-NonUniform*, agents are additionally penalized if their actions are not in the direction of their target. In each case, F_t^i is positive when agents take actions that move it towards their target. $\alpha., \beta., M.$ are constants.

In our experiments, we *anchor* each agent to a (distinct) landmark. The shaping advice in this case will depend on the distance of the agents to the landmarks they are anchored to. Although distances computed in this manner will depend on the order in which the agents and landmarks are chosen, we observe that it empirically works across multiple training episodes where positions of landmarks and initial positions of agents are generated randomly. The advice provided by SAM is positive when agents moves closer to the landmark they are anchored to. This ensures that the agents cover the landmarks. In the absence of anchoring, they may get distracted to move towards different landmarks at each time step. This phenomenon is observed during transitions between episodes, when positions of agents, landmarks, and the identity of the true landmark (in physical deception) are reset. It is also seen when the closest landmark to an agent at time t may be different from that at time $t + 1$. Anchoring results in agents learning to cover landmarks faster.

We consider two variants of advice for each task. In ***SAM-Uniform***, the advice for

every action taken in a state is the same. In ***SAM-NonUniform***, a higher weight is given to some 'good' actions over others at each state. We test the utility of providing advice on specific actions when different numbers of agents are present in the cooperative navigation and physical deception tasks. We compare the performance of agents trained with SAM (SAM-Uniform or SAM-NonUniform) augmented to MADDPG against a baseline where agents are trained with classical MADDPG policies. We enumerate this form of advice for each task in Table 4.1.

4.3.3 Architecture and Training

During training, we collect agents' experiences in a replay buffer, $\mathcal{B}$. $\mathcal{B}$ contains a collection of tuples of the form $\{(s_t, a_t^1, \ldots, a_t^n, r_t^1, \ldots, r_t^n, s_{t+1})\}$. The procedures to update the parameters of the actor and critic for each agent are carried out by sampling a set of experiences from $\mathcal{B}$. The centralized critic allows sharing of parameters during training, which enables updating agents' policies in parallel.

We train with five random seeds for all tasks, and adopt the same network architectures and training configurations as used in [55] when comparing cases where agents are equipped with SAM and those where agents use classical MADDPG policies. In the competitive tasks, we evaluate agents equipped with SAM against adversaries who use classical MADDPG policies.

4.3.4 Results

Figure 4.3 shows $0-1$ normalized scores, averaged over the last 1000 training episodes, comparing SAM augmented to MADDPG and classical MADDPG policies. The **score** for a task is the average agent reward in cooperative tasks, and the *average agent advantage* (= agent − adversary reward) in competitive tasks [92]. Agents equipped with *SAM-NonUniform* have the best performance. This is because SAM-NonUniform provides specific feedback on the quality of agents' actions. *SAM-Uniform* also performs well in these tasks. SAM-NonUniform outperforms SAM-Uniform and the classical MADDPG baseline by a significant

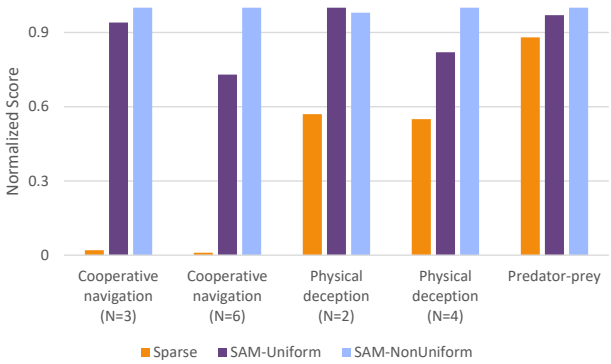


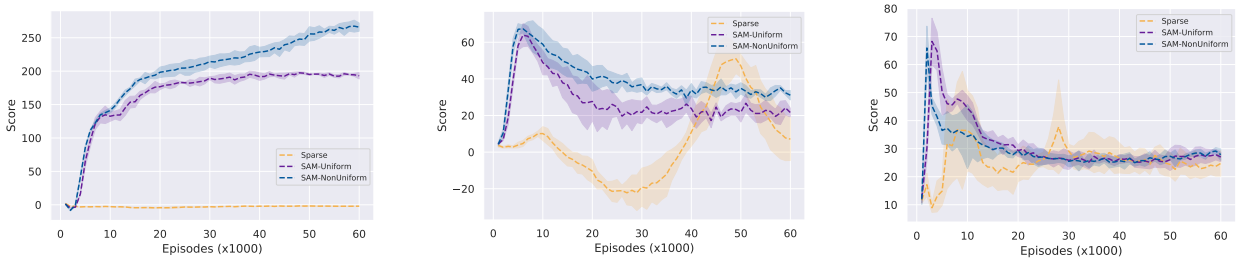
Figure 4.3: Comparison between SAM (SAM-NonUniform (blue) or SAM-Uniform (purple)) augmented to MADDPG and classical MADDPG policies (orange) on cooperative and competitive tasks with sparse rewards. The **score** for a task is the average agent reward in cooperative tasks, and the *average agent advantage* (= agent reward − adversary reward) in competitive tasks. Each bar cluster shows Normalized $0-1$ scores, averaged over the last 1000 training episodes. Higher score is better. SAM-NonUniform outperforms SAM-Uniform and the classical MADDPG baseline by a larger margin when there are more agents in the cooperative navigation and physical deception tasks.

margin when there are more agents.

In cooperative navigation, when rewards are sparse, the agents are not able to learn policies that will allow them to even partially cover the landmarks using the MADDPG baseline. In comparison, SAM guides agents to learn to adapt to each others' policies, and cover all the landmarks. SAM-NonUniform results in much higher rewards than other methods in the complex task with $N = 6$ agents.

We observe a similar phenomenon in physical deception, where SAM guides agents to learn policies to cover the landmarks. This behavior of the agents is useful in deceiving the adversary from moving towards the true landmark, thereby resulting in lower rewards for the adversary. Therefore, agent advantage is higher with SAM.

In the predator-prey task, we see that the performance of SAM is comparable to MAD-



(a) CN ($N = 6$) (b) PD ($N = 4$) (c) PP (3 pred., 1 prey.)

Figure 4.4: Average scores and variance when agents use SAM-NonUniform (blue), SAM-Uniform (purple), and classical MADDPG (orange) policies for tasks in the Particle World environment that have sparse rewards. SAM-NonUniform results in the highest average scores. SAM-Uniform compares favorably, and both significantly outperform classical MADDPG policies. This is especially evident during early stages of training. In the cooperative task, only SAM is able to guide agents to cover all landmarks. In a sparse reward setting, without SAM, the agents are not able to learn policies that will enable them to cover the landmarks. In competitive tasks, SAM ensures that agent policies adapt better to each other. This is shown by oscillations of smaller amplitude in the average score with SAM than without.

DPG. We believe that this is because this task might have a well-defined and unique *equilibrium* to which agent policies eventually converge.

Figure 4.4 shows the average and variance of the score during different stages of the training process. We observe that *SAM-NonUniform* and *SAM-Uniform* allow agents to obtain higher scores in the early stages of training. These two methods dominate the MADDPG baseline. The improved performance of SAM in early stages of training is also evident in predator-prey. Although SAM encourages predators to take actions towards the prey, this could make the prey exit the screen. Since the reward structure penalizes the prey for this action, it results in a higher score on the task. However, as training progresses, the prey adapts its behavior to learn policies to avoid this action. As a consequence, the score reduces during later stages of training.

The authors of [55] observed that agent policies being unable to adapt to each other in competitive environments resulted in oscillations in rewards. Figure 4.4b indicates that SAM is able to alleviate this problem. In the sparse reward physical deception environment, policies learned by the agents using SAM result in much smaller oscillations in the rewards than when using MADDPG.

4.4 Summary

This chapter presented SAM, a framework to incorporate domain knowledge in mixed cooperative-competitive MARL. SAM used a centralized critic to learn decentralized policies. Shaping advice provided to each agent during training helped address the multi-agent credit assignment problem. This advice was a heuristic that was specified as a difference of potential functions, and gave an agent immediate feedback on the quality of its actions. Agents equipped with SAM did not get distracted from accomplishing task objectives specified by the environment reward. We established theoretical guarantees of convergence of policy gradients and values when using SAM. We empirically showed that SAM accelerated learning of policies, and resulted in improved agent performance in three *tasks with sparse rewards in the Particle World environment*. In competitive tasks, SAM alleviated a problem of 'os-

cillating rewards'. We observed that agent policies adapted better to each other with SAM than without, resulting in more stable rewards.

Chapter 5

INTERACTIVE REWARD SHAPING USING HUMAN FEEDBACK

5.1 Motivation

Although RL algorithms have shown impressive results in certain environments [61, 75], humans are usually much more efficient in terms of the number of actions required to obtain higher rewards. This has especially been observed in environments with high-dimensional state spaces, like video games, where states are raw pixels of images or snapshots of videos. The authors of [67] showed that a human player is easily able to play and win games in setups where the reward structure is sparse or significantly delayed (for e.g. receiving a reward only for successfully completing one level of a game), while deep RL algorithms struggle. In these settings, any prior knowledge that a human might have about the environment, and their ability to learn from the environment, is crucial in determining success. In some cases, the domain knowledge may not be in the form of potential functions but human operators are available to provide feedback during training. The role of a human operator in providing a *shaping* reward signal to aid the learning process of the RL agent in such environments has not been addressed in prior work.

In this chapter, we seek to effectively integrate human feedback with deep RL algorithms in high-dimensional state spaces. We term this ***FRESH***, for **Feedback-based REward SHaping**. We assume that there is a human operator who can provide feedback on actions taken by the RL agent. During training, the operator is presented with trajectories (sequences of states and actions) from a replay buffer and indicates whether an action at a state in the trajectory is *good* or *bad*. The operator is additionally able to provide this feedback on the states. They can also indicate that they are not sure if an action is good or

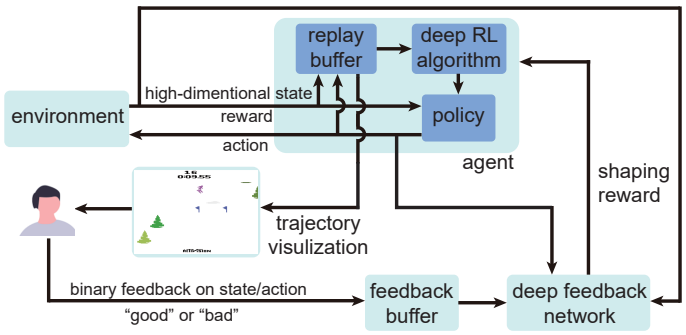


Figure 5.1: Schematic for *FRESH (Feedback-based REward SHaping)*. A human operator is presented with trajectories of game-play from the replay buffer. At each state in the trajectory, the operator indicates whether the action taken in that state is *good* or *bad*. At some states, the operator also indicates whether the state is *good* or *bad*. This feedback is stored in a feedback buffer. A deep feedback neural network is used to allow the deep reinforcement learning algorithm to generalize feedback signals obtained during training to unseen states and actions at test-time. The output of the feedback network is converted to a shaping reward, which is augmented to the reward provided by the environment.

bad (*cannot tell*). If they feel that sufficient number of feedback signals have been given on a trajectory, they can choose to *skip* the remainder of that trajectory. The motivation for this type of feedback is that it is relatively easier for a human to understand whether a state and the consequence of an action taken at a state will be good or bad. A schematic of our setup is shown in Figure 5.1.

We wish to clarify that this approach is different from that used in Deep-TAMER [91]. Deep-TAMER used human-feedback, but did not use the environment reward, and instead adopted a supervised learning framework during training. In [91], a human operator associated a numerical value indicative of how good the agent's behavior was, in the opinion of the operator. In comparison, in *FRESH*, the operator only needs to provide a qualitative indication of whether an action in a state is good or bad, and we convert this to a shaping reward that is integrated with the reward from the environment.

Due to the size of the state space when operating in high dimensional state spaces, it will not be uncommon for the agent to encounter states at test time that it would have never seen during training. Moreover, the number of feedback signals that can be provided by a human is limited. Therefore, we need to be able to generalize from the feedback in order to determine whether a state encountered at test time is 'similar' to some state seen previously. Neural networks (NNs) will allow us to generalize from feedback. Since an NN yields an output for any input given to it, we require a mechanism to reject outputs that could lead the RL agent towards 'bad' states. One way to achieve this is by quantifying a measure of confidence of the network in its output. This way, by setting an appropriate threshold, outputs of the network that have a confidence score below this threshold can be rejected. Only those outputs with a confidence score above the threshold will be retained and used during the training phase. The authors of [89] used the output of a softmax function as a measure of confidence. However, a shortcoming of this approach is that a model could be uncertain in its predictions even with a high softmax output [25].

Bayesian neural networks [63] have been used to represent the uncertainty in the output of an NN. Two techniques that offset the high costs incurred during inference are bootstrapped ensemble NNs [45, 66] and dropout as Bayesian approximation [25]. The bootstrap principle is to approximate the distribution of a population by a sample distribution [20]. This property allows us to use an ensemble of NNs to effectively represent uncertainty in the model where each network in the ensemble produces a value indicative of its confidence in its output. In the dropout approach, an equivalence was established between *dropout* training in deep NNs and Bayesian inference in Gaussian processes. We prefer the bootstrapped ensemble over dropout in this chapter because of its lower time complexity during inference.

5.2 Feedback-based Reward Shaping

The use of human feedback in high-dimensional state spaces raises the following questions:

Q1: How can feedback signals provided by a human operator be effectively used in high-

dimensional state spaces?

Q2: How can this feedback be meaningfully integrated with deep reinforcement learning (DRL) algorithms?

In this section, we present ***FRESH***, a feedback-based reward shaping approach towards answering these questions. We assume that an RL agent has to learn to maximize its reward in an environment that has a high-dimensional state space. An example of such an environment is an Atari game in the Arcade Learning Environment. 'States' in this environment are collections of pixels (images). Although deep NNs have been successful in outperforming a human expert player in many Atari games [61], there are some games where the expert is still able to perform better than state-of-the-art DRL algorithms. This chapter studies two examples of the latter (Bowling and Skiing), and we demonstrate that *FRESH* performs at least as well as a human expert player in these environments.

5.2.1 Binary-valued Human Feedback

We assume that a human operator is able to provide binary-valued feedback on actions and states. We do not require the operator to be an expert. However, we assume that this operator will have the ability to understand game-play in the environment after explanation by an expert. Feedback given on an action is a (local) interpretation of the quality of the action at a particular state, independent of how 'good' or 'bad' the state is. In comparison, feedback provided on a state is a (more global) interpretation on whether the current observed state is good or bad in terms of the rewards that can be obtained. The reason we use this type of feedback signal is that in many settings, rather than assigning a numeric value, it is relatively easier for a human to interpret whether a state or an action taken in a particular state is simply 'good' or 'bad'. Specifically, the human provides feedback on actions implicitly according to a hidden function $h_{s,a}(\cdot,\cdot) : S \times A \to \{0,1\}$, and feedback on states according to a hidden function $h_s(\cdot) : S \to \{0,1\}$. In each case, 0/1 denote *bad/ good*. The human operator can additionally provide feedback signals that indicate *not sure* (if they are not

certain whether an action in a state is 'good' or 'bad') and *skip* (if the operator feels that sufficient number of feedback signals have been provided for a trajectory). These latter two signals are not used to train the agent, but they allow the human operator to significantly reduce the amount of time spent in the training phase. We note that both $h_{s,a}$ and h_s can be time-varying, implying that the feedback can change for the same state or state-action pair as the agent training process proceeds. Moreover, the human operator might also change their expectations of the agents' performance over time, which justifies the time-varying nature of the feedback signal.

We assume that at each time, in a state s_t, there is exactly one action a_t that is the 'best' in that state. That is, taking this action will lead to the agent receiving a higher (accumulated) reward than taking any other action. We formulate a binary classification problem, and use maximum likelihood estimation (MLE) to determine the best action in a state. We denote the predicted probability that action a_i is the best in state s by $f^{a_i}(s)$. Furthermore, $\sum_{i=1}^{|A|} f^{a_i}(s) = 1$. To implement our maximum likelihood estimator, we use cross-entropy loss with both positive and negative labels:

$$\mathcal{L}(f^{a_i}(s), h_{s,a_i}) = -h_{s,a_i} \log f^{a_i}(s) - (1 - h_{s,a_i}) \log \sum_{a \neq a_i} f^{a}(s),$$

where h_{s,a_i} is the binary-valued human feedback associated to state-action pair (s, a_i).

We formulate an analogous classification problem for feedback on states. Let $g(s)$ denote the predicted probability that state s is *good*. The loss function used to evaluate this prediction is:

$$\mathcal{L}(g(s), h_s) = -h_s \log g(s) - (1 - h_s) \log(1 - g(s)),$$

5.2.2 Generalizing Human Feedback

A challenge in high-dimensional state spaces that is not encountered in the tabular setting is that the agent might observe a lot of states at test-time that it might have never seen during training. Moreover, since the number of feedback signals that a human can provide is

often limited, it is important to be able to *generalize* from the feedback in order to determine whether a state encountered at test time is similar to some state seen during training. In order to fully exploit the feedback signals given by a human, we leverage deep neural networks (DNNs). DNNs have been shown to have the ability to use feedback on states and actions and generalize it to other states and actions that have not been previously seen, but 'similar' to those already known [27]. We term the neural networks used to abstract the feedback provided by a human operator as *feedback neural networks (FNNs)* for the rest of this chapter.

Using NNs to generalize human feedback presents another challenge. NNs typically produce an output for any input, and do not have a measure of confidence. This could lead to a scenario when the network produces an arbitrary output to an input state that it has never seen during training, and this could lead the RL agent to undesired or incorrect states. This necessitates development of a mechanism that allows the agent to reject the output of FNNs.

A class of NNs called *Bayesian neural networks* [63] are able to produce both an output, and a value indicating the confidence of the NN in the output. However, this process is computationally expensive. To obtain these confidence values, approximations of Bayesians NNs using dropout [25] and ensemble of NNs for predicting model uncertainty [45, 66] have been proposed. We use the bootstrap ensemble NN architecture in this chapter due to its lower time complexity during inference when compared to dropout.

In order to achieve a further reduction in complexity, we use a shared network architecture as shown in Figure 5.2. In this architecture, the entire network has $K_s + K_a$ heads, with K_s bootstrapped heads for learning feedback on states and K_a bootstrapped heads for learning feedback on actions, along with a shared network. Each head is randomly initialized and trained on a bootstrapped subset of feedback data, that is sampled with replacement from the entire feedback data. All heads share the same preceding layers, which allows the entire network to be trained more efficiently. We collect the prediction from each head and use the mean of these predictions as the final prediction.

Let $\mathbf{f}_j = \left(f_j^{a_1}(s), f_j^{a_2}(s), \ldots, f_j^{a_{|A|}}(s)\right)$ denote the output of the j-th action head, where $f_j^{a_i}(s)$ is the probability that action a_i is the best in state s, and let $g_j(s)$ denote the output

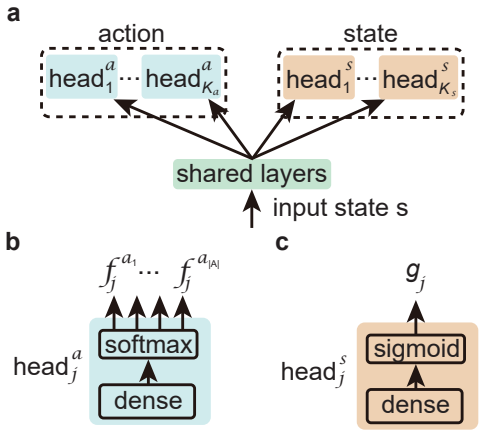


Figure 5.2: Figure **a** shows the multi-head architecture with shared layers for the feedback neural network that we use in this chapter. Figures **b** and **c** show architectures of parts of the networks corresponding to learning feedback on actions and feedback on states respectively.

of the j-th state head. In order to predict the human feedback on actions, we define:

$$\text{pred}_{\text{action}}(s) = \underset{i \in \{1,\ldots,|A|\}}{\arg\max} \left\{ \frac{1}{K_a} \sum_{j=1}^{K_a} f_j^{a_i}(s) \right\} \tag{5.1}$$

and to predict human feedback on states, we define:

$$\text{pred}_{\text{state}}(s) = \begin{cases} 1, & \text{if } \dfrac{1}{K_s} \displaystyle\sum_{j=1}^{K_s} g_j(s) > 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

We calculate the empirical standard deviation of individual prediction by different heads and use it as a proxy for the confidence value. Let $p_j^a(s) = \underset{i \in \{1,\ldots,|A|\}}{\arg\max} f_j^{a_i}(s)$ denote the prediction of the j-th action head. Then, the confidence value $c_a(s)$ for predicting human feedback on actions can be obtained by computing the empirical standard deviation:

$$c_a(s) = 1 - \sqrt{\frac{1}{K_a - 1} \sum_{j=1}^{K_a} \left(p_j^a(s) - \frac{\sum_{j=1}^{K_a} p_j^a(s)}{K_a} \right)^2} \tag{5.3}$$

The (proxy for the) confidence $c_s(s)$ of predicting the human feedback on states can be computed similarly.

5.2.3 Reward Shaping

The reward signal supplied by the environment can often be sparse and/ or significantly delayed, although it can *accurately* define desired goals for the agent. In order to make the learning procedure more efficient, the human feedback can be introduced through reward shaping. Once an approximation of the human feedback model is available, we can transfer the model output to more frequent and timely reward signals, although this additional reward may sometimes be incorrect due to an error made by human operator providing the feedback.

In order to make use of the feedback given by the human operator, and not limit the agent's performance when this operator makes an error, we present a way to combine FNNs with the environment reward. Let s_t, a_t, and s_{t+1} respectively denote the state and action at time t, and the next state after taking action a_t in s_t. The feedback received on actions is transferred to an additional reward r_a as:

$$r_a(s_t, a_t, s_{t+1}) = \begin{cases} 1, & \text{if } a_t = \text{pred}_{\text{action}}(s_t) \text{ and } c_a(s_t) > 1 - \beta_a \\ 0, & \text{otherwise,} \end{cases} \tag{5.4}$$

where β_a is a pre-assigned constant threshold. If the action taken by the agent is consistent with the best action predicted by the FNN and the confidence of FNN in the prediction $c_a(s_t)$ is higher than $1 - \beta_a$, then an additional reward is provided. Similarly, feedback received on states can be transferred to an additional reward r_s as:

$$r_s(s_t, a_t, s_{t+1}) = \begin{cases} 1, & \text{if } \text{pred}_{\text{state}}(s_{t+1}) = 1 \text{ and } c_s(s_{t+1}) > 1 - \beta_s \\ 0, & \text{otherwise,} \end{cases} \tag{5.5}$$

If r_e denotes the environment reward, the *shaped reward* $r(s_t, a_t, s_{t+1})$ is then:

$$r(s_t, a_t, s_{t+1}) = r_e(s_t, a_t, s_{t+1}) + \lambda_a r_a(s_t, a_t, s_{t+1}) + \lambda_s r_s(s_t, a_t, s_{t+1}), \tag{5.6}$$

where λ_a and λ_s are the weights, balancing the importance of the three rewards. Although λ_a and λ_s can decay over time, during our experiments we observed that keeping them constant works well.

5.2.4 Algorithm

We evaluate *FRESH* when it is used with deep RL algorithms that use a replay buffer for learning, e.g. DQN [61]. The procedure for collecting the human feedback is summarized in

Algorithm 3 HumanFeedbackCollection

Input: Replay buffer B_q storing trajectory experience and buffer B_f storing human feedback. Masking distributions M_s, M_a

1: Sample a trajectory τ from B_q and visualize τ for feedback.
2: **for** $(s_t, a_t, r_t, s_{t+1}) \in \tau$ **do**
3: **if** new feedback on state f_s is available **then**
4: sample $\mathbf{m}_s \sim M_s$ and store $(s_{t+1}, f_s, \mathbf{m}_s)$in B_f
5: **end if**
6: **if** new feedback on action f_a is available **then**
7: sample $\mathbf{m}_a \sim M_a$ and store $(s_t, a_t, f_a, \mathbf{m}_a)$ in B_f
8: **end if**
9: **end for**
10: **return** B_f

Algorithm 5. First, we sample a trajectory τ from the replay buffer B_q (line 1). Any choice of sampling method can be used in this step. For example, the trajectory with highest or lowest reward may be given higher priority at different training stages. This sampled trajectory is then presented to the human operator in order to receive feedback. The speed at which τ is displayed to the human can be much slower than actual game play. For example, when the states are represented by images, we can apply a lower frame rate so that it will be easier for the operator to assess states and actions in the trajectory. After feedback on a state f_s or feedback on an action f_a is provided, a mask $\mathbf{m}_s \in \mathbb{Z}^{K_s}$ or $\mathbf{m}_a \in \mathbb{Z}^{K_a}$ will be sampled from a masking distribution. This will indicate which heads this feedback should be used to train. For example, the components of the mask can be independently drawn from

a Bernoulli distribution (double or nothing bootstrap) or from an exponential distribution (Bayesian non-parametric posterior of a Dirichlet process) [66]. We note that feedback will not be provided on all states or actions, since the human operator might refuse to provide an assessment if they are not sure of the quality of the state/action. The feedback together with the mask will be stored in the feedback buffer (line 3-8).

Algorithm 4 describes *FRESH*. The feedback buffer is initialized by providing feedback on trajectories from random play to train the feedback networks FNN (lines 1-7). The FNNs are updated using stochastic gradient descent (SGD). For training the value networks Q, we use a DQN-based algorithm [87, 90], but the reward function is changed to Equation (5.6) (lines 9-12). The human operator is asked to provide feedback every N_c episodes (lines 13-15). If N_f new feedback signals are available, the FNN is re-tuned (lines 16-19).

5.3 Experiments

We evaluate *FRESH* on two Atari games in the Arcade Learning Environment [3]. Figure 5.3 shows screen shots of game play in the *Bowling* and *Skiing* environments. Although human experts can achieve high scores with relative ease in these games, it has been extremely difficult for state-of-the-art deep reinforcement learning (DRL) algorithms to match this. We observe that the performance on these games using *FRESH* compares favorably with that of a human expert, and is significantly improved over other DRL algorithms.

5.3.1 Game Description and Experiment Setup

The Bowling game comprises four actions, *no-action, up, down, fire.* A game lasts 10 rounds and the player (agent) gets two chances in each round to roll the bowling ball to knock down as many pins as possible. The game begins with the player choosing a position to release (*fire*) the ball by moving vertically using actions *up* or *down.* After releasing the ball, the player gets one chance to steer the direction of the ball by taking actions *up* or *down.* The reward structure of the game makes it difficult for state-of-the-art deep reinforcement learning (DRL) algorithms to obtain a high reward. In particular, the player does not receive

Algorithm 4 *FRESH* for DQN

Input: Value networks Q. Feedback networks FNN. Masking distributions M_s, M_a. Replay buffer B_q storing experience for training DQN and buffer B_f storing human feedback for training FNN. Thresholds β_a and β_s. Weights λ_a and λ_s. Feedback collection frequency N_c and FNN update frequency N_f. Initial number of trajectories and feedback n_i and m_i.

repeat
  Sample trajectories based on random play and store trajectories in B_q.
until Collect n_i trajectories in B_q
repeat
  $B_f = HumanFeedbackCollection(B_q, B_f, M_s, M_a)$
until collect m_i feedback in B_f
sample batches from B_f and update FNN using SGD
for Episode i=1, ... **do**
  repeat
    execute action $a = \arg\max_a Q(s, a)$, observe reward r and next state s' and store (s, a, r, s') in B_q
    update Q using DQN algorithm but change reward function to Equation (5.6)
  until end of episode
  if i % $N_c == 0$ **then**
    $B_f = HumanFeedbackCollection(B_q, B_f, M_s, M_a)$
  end if
  if Number of new feedback $new_f > N_f$ **then**
    sample batches from B_f and update FNN using SGD
    $new_f \leftarrow 0$
  end if
end for

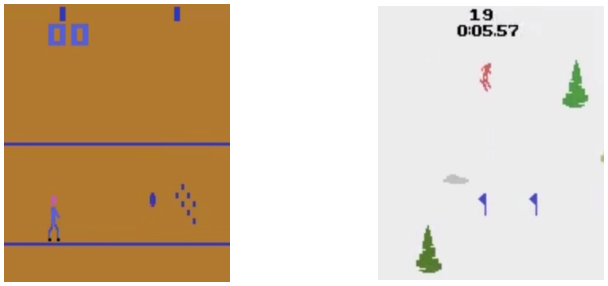


Figure 5.3: Snapshots from game-play in the Atari games of Bowling (left) and Skiing (right). In Bowling, the goal for the player is to roll the ball and knock down as many pins as possible. In Skiing, the goal for the player is to reach the bottom of a valley as soon as possible, and at the same time, pass through as many gates as possible while avoiding obstacles.

an immediate reward if all pins are knocked down in one turn. Instead, this reward will be supplied at the end of the next turn, together with the reward for that turn. This makes it difficult for DRL algorithms to correctly provide credit for actions by simply looking at the reward.

The Skiing game consists of three actions, *no-action, left, right*. The player controls their direction of motion in order to avoid obstacles (trees and moguls) and pass through the gates. The goal for the player is to reach the bottom of the valley as soon as possible, and in the process, pass through as many gates as possible. This game is difficult for DRL algorithms to play since the reward indicating the number of gates passed through is supplied only at the end the game, making the credit assignment task difficult.

In all our experiments, we use the same neural network architecture and hyper-parameters for DQN as [90] does. Additionally, we apply double Q-learning [87] to avoid overestimation of action values. The shared network of our feedback neural network (FNN) has the same architecture as the convolutional layers of DQN, and each head of the FNN adopts three fully-connected layers with batch-normalization. Each state is a tensor stacked by 4 gray-scale images, obtained by converting 4 consecutive colored video game frames. The regions of the frames showing the game score are removed when the frame is an input to the FNN.

We clip the environment reward using the same approach as [61] does, and set $\lambda_a = 0.2$ and $\lambda_s = 0.1$ across the experiments. In the early stages of training the FNN, when sufficient diverse feedback signals are not available, the agent might be distracted from the true goal and run into a *cycling* problem, as indicated in [64]. To alleviate this problem, when $|s_t - s_{t-1}|$ is smaller than a known threshold value (which is indicative of the agent being stuck in a local state), we give the agent a penalty to offset the reward supplied by the FNN. This works well in our experiments. Bernoulli distribution is used as the masking distribution. To evaluate Algorithm 4, a human operator provides feedback using a computer with an user interface to visualize trajectories and receiving feedback. While providing feedback, the human trainer is allowed to say *not sure* for states and actions for which the human is not sure about the quality. In the following, we first evaluate the performance of *FRESH* in both Bowling and Skiing and then present a detailed study on the effect of the different components of the FNN in Bowling.

5.3.2 FRESH *for Bowling and Skiing*

We first collect $n_i = 100$ trajectories using random play, which are sorted based on the accumulated reward. The trajectory with the highest reward is given the highest priority to obtain human feedback. We collect a total of $m_i = 500$ feedback signals for Bowling and $m_i = 5000$ signals for Skiing from trajectories of random play. While training the DQN, we collect feedback signals every $N_c = 30$ episodes. When $N_f = 300$ new feedback signals is available, we update the FNN using all the data in the feedback buffer. In our experiments, we observed that the ratio of the feedback signal *good* over *bad* is around 0.2. In this section, we fix the number of heads $K_a = K_s = 10$ and thresholds ($\beta_a = 1.0, \beta_s = 0.02$).

Figure 5.4 show the performance of *FRESH* in *Bowling* and *Skiing*. Each curve is an average over five runs with different human feedback and initialization. Shaded regions indicate the standard deviation. We compare the performance of *FRESH* (Algorithm 4) with Double-DQN [87], IMPALA [21], Rainbow [34], and Human expert. We use the best final performance reported in the literature for Double-DQN, IMPALA and Rainbow. The

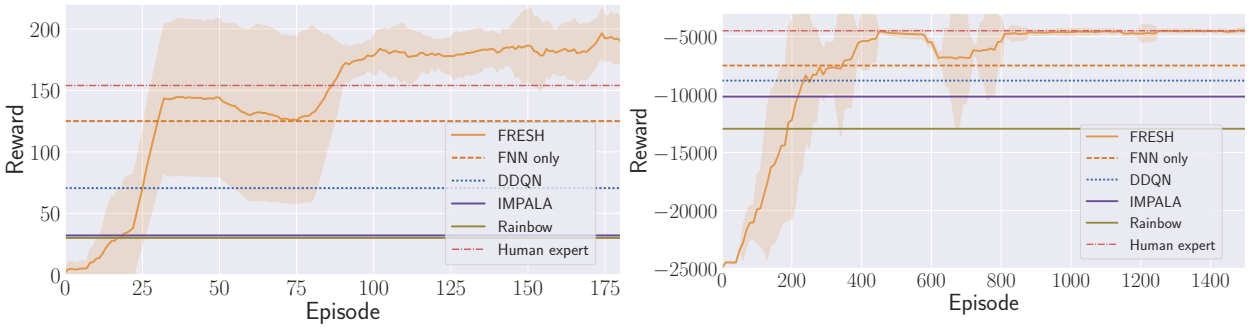


Figure 5.4: In Bowling (left), *FRESH* outperforms state-of-the-art deep reinforcement learning algorithms, and also outperforms a human expert player. In Skiing, *FRESH* outperforms state-of-the-art deep reinforcement learning algorithms, and performs as well as a human expert player. The shaded region indicates the variance of the reward.

human expert performance data is from [61]. Since the trained FNN is able to output an estimated best action for every state, it can also be used as a policy. Therefore, we report the performance of using FNN alone as well.

FRESH allows the agent to learn good policies in both environments. In contrast, state-of-the-art DRL algorithms fail even after training for $>$ 100 million frames. *FRESH* lets the agent learn policies that allow it to obtain higher scores than a human expert in *Bowling*. This indicates that *FRESH* can not only guide the learning process but also that the final performance is not limited by the quality of feedback. In *Skiing*, *FRESH* outperforms the policy induced by the FNN and achieve a similar score as a human expert.

We observe that *FRESH* allows the agent to achieve an average score of **187** in *Bowling*. This is **21.4**% higher than the average score obtained by a human expert in this environment. The highest score obtained by our algorithm in *Bowling* is $>$ **200**. State-of-the-art deep RL algorithms perform poorly in comparison- DDQN obtains an average score of 70.5, while the other methods report an even lower score. Deep-TAMER [91], on the other hand, is able to achieve an average score of around 180, and a high-score of around 200. In *Skiing*, *FRESH* is able to obtain an average score of $-$**4400**, which is equal to that obtained by

an human expert player. The deep RL algorithms perform poorly in this environment also. The authors of [91] do not report a score in the Skiing environment.

5.4 Summary

We presented *FRESH*, a feedback-based reward shaping framework to effectively integrate human feedback with deep RL algorithms in high-dimensional state spaces. We used a feedback neural network to effectively generalize feedback signals provided by the human operator, and an ensemble of neural networks to represent the confidence of the neural network in its output. Our approach was evaluated on the *Bowling* and *Skiing* Atari games of the arcade learning environment. *FRESH* performed much better than state-of-the-art deep learning algorithms in these environments. In *Bowling*, *FRESH* obtained an average score of $\mathbf{187}$, which was $\mathbf{21.4}\%$ higher than the score obtained by a human expert [61]. The highest score obtained by *FRESH* in Bowling was $> \mathbf{200}$. In *Skiing*, *FRESH* obtained an average score of $-\mathbf{4400}$, which was equal to that obtained by a human expert player.

Chapter 6

AGENT-TEMPORAL ATTENTION FOR REWARD SHAPING IN MULTI-AGENT REINFORCEMENT LEARNING

6.1 Motivation

Previous chapters have discussed reward shaping approaches based on either potential functions or human feedback. In this chapter, we aim to scale up reward shaping to settings with multiple agents, where neither domain knowledge in terms of potential functions, nor human operators may be available. Specifically, we consider multi-agent reinforcement learning (MARL) tasks where agents receive a shared global reward at the end of an episode. Without prior domain knowledge, we propose an algorithm which can automatically perform effective long-term temporal credit assignment based on the interaction history of agents.

The long-term temporal credit assignment problem has been studied in single-agent RL by performing return decomposition via contribution analysis [2] and using sequence modeling [52]. These methods do not directly scale well to MARL since size of the joint observation space grows exponentially with number of agents [55]. Besides scalability, addressing temporal credit assignment in MARL with episodic rewards presents two challenges. It is critical to identify the relative importance of: i) each agent's state at any single time-step (*agent dimension*); ii) states along the length of an episode (*temporal dimension*). We introduce *Agent-Temporal Attention for Reward Redistribution in Episodic Multi-Agent Reinforcement Learning (AREL)* to address these challenges.

AREL uses attention mechanisms [88] to carry out *multi-agent temporal credit assignment* by concatenating: i) a *temporal attention module* to characterize the influence of actions on state transitions along trajectories, and; ii) an *agent attention module*, to determine how any single agent is affected by other agents at each time-step. The attention modules enable

learning a redistribution of the episodic reward along the length of the episode, resulting in a *dense* reward signal. To overcome the challenge of scalability, instead of working with the concatenation of (joint) agents' observations, *AREL* analyzes observations of each agent using a temporal attention module that is shared among agents. The outcome of the temporal attention module is passed to an agent attention module that characterizes the relative contribution of each agent to the shared global reward. The output of the agent attention module is then used to learn the redistributed rewards.

When rewards are delayed or episodic, it is important to identify 'critical' states that contribute to the reward. The authors of [26] recently demonstrated that rewards delayed by a long time-interval make it difficult for temporal-difference (TD) learning methods to carry out temporal credit assignment effectively. AREL overcomes this shortcoming by using attention mechanisms to effectively learn a redistribution of an episodic reward. This is accomplished by identifying critical states through capturing long-term dependencies between states and the episodic reward.

Agents that have identical action and observation spaces are said to be *homogeneous*. Consider a task where two homogeneous agents need to collaborate to open a door by locating two buttons and pressing them simultaneously. In this example, while locations of the two buttons (states) are important, the identities of the agent at each button are not. This property is termed *permutation invariance*, and can be utilized to make the credit assignment process sample efficient [26, 52]. Thus, a redistributed reward must identify whether an agent is in a 'good' state, and should also be invariant to the identity of the agent in that state. *AREL* enforces this property by designing the credit assignment network with permutation-invariant operations among homogeneous agents, and can be integrated with MARL algorithms to learn agent policies.

6.2 Attention-based Reward Shaping

We consider MARL tasks where agents share the same global reward, which is received only at the end of an episode. The objective is to redistribute this episodic reward for

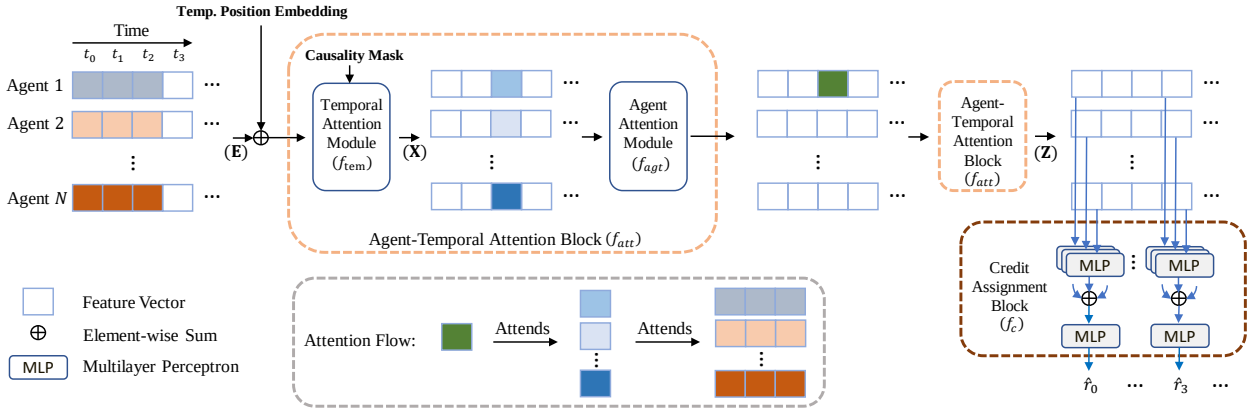


Figure 6.1: Schematic of AREL. The agent-temporal attention block concatenates temporal and agent attention modules, and summarizes input feature (e.g. observation) vectors. This is accomplished by establishing relationships between (*attending to*) information along time and among agents. The attention flow indicates that an output feature vector of the agent-temporal attention block for an agent at a time t (green square) can attend to input features from all other agents before and including time t. Multiple agent-temporal attention blocks can be concatenated to each other to improve expressivity. The output of the last such block is fed to the credit assignment block, which applies shared multi-layer perceptrons to each attention feature. The output is the redistributed reward, which is integrated with MARL algorithms (e.g. MADDPG, QMIX) to learn agent policies.

effective multi-agent temporal credit assignment. To accomplish this goal, it is critical to identify the relative importance of: i) individual agents' observations at each time-step, and; ii) observations along the length of a trajectory. We introduce *AREL* to address the above challenges. *AREL* uses an agent-temporal attention block to infer relationships among states at different times, and among agents. A schematic is shown in Fig. 6.1, and we describe its key components and overall workflow in the remainder of this section.

6.2.1 Decentralized Partially Observable Markov Decision Process

A fully cooperative multi-agent task will be specified as a decentralized partially observable Markov decision process (Dec-POMDP) [65], which is a special case of stochastic game. A Dec-POMDP is a tuple $G = (S, A, P, r, Z, O, n, \gamma)$, where $s \in S$ describes the environment state. Each agent $i \in \{1, 2, \ldots, n\}$ receives an observation $o^i \in O^i$ according to an observation function $Z(s, i) : S \times \mathbb{N} \to O$. At each time step, agent i chooses action $a^i \in A^i$ according to its policy $\pi^i : O^i \times A^i \to [0, 1]$. $A^1 \times \cdots \times A^n := A$ forms the joint action space, and the environment transitions to the next state according to the function $P : S \times A^1 \times \cdots \times A^n \to S$. All agents share a global reward $r : S \times A \to \mathbb{R}$. The goal of the agents is to determine their individual policies to maximize the *return*, $J := \mathbb{E}_{s\sim P, a^1\sim\pi^1,\ldots,a^n\sim\pi^n}[\sum_{t=0}^{T} \gamma^t r_t(s, a^1, \ldots, a^n)]$, where γ is a discount factor, and T is the length of the horizon. Let $a_t := (a_t^1, \ldots, a_t^n)$ and $R_t := \sum_{l=0}^{T-t} \gamma^l r_{t+1}$. A trajectory of length T is an alternating sequence of observations and actions, $\tau := (o_0, a_0, o_1, a_1, \ldots, o_T)$.

6.2.2 Agent-Temporal Attention

In order to redistribute an episodic reward in a meaningful way, we need to be able to extract useful information from trajectories. Each trajectory contains a sequence of observations involving all agents. At each time-step of an episode of length T, a feature of dimension D corresponds to the embedding of a single observation. When there are N agents, a trajectory is denoted by $\mathbf{E} \in \mathbb{R}^{T\times N\times D}$. The objective is to learn a mapping $f_{arel}(\mathbf{E}) : \mathbb{R}^{T\times N\times D} \to \mathbb{R}^T$ to assign credit to the agents at each time-step. The information in a trajectory $\mathbf{E}$ comprises

two parts: (1) *temporal information* between (embeddings of) observations at different time steps: this provides insight into the influence of actions on transitions between states; (2) *structural information*: this provides insight into how any single agent is affected by other agents.

These two parts are coupled, and hence studied together. The process of learning these relationships is termed *attention*. We propose an *agent-temporal attention structure*, inspired by the Transformer [88]. This structure selectively pays attention to different types of information- either from individual agents, or at different time-steps along a trajectory. This is accomplished by associating a weight to an observation based on its relative importance to other observations along the trajectory. The agent-temporal attention structure is formed by concatenating one agent attention module with one temporal attention module. The temporal attention modules determine how entries of $\mathbf{E}$ at different time-steps are related (along the 'first' dimension of $\mathbf{E}$). The agent attention module determines how agents influence one another (along the 'second' dimension of $\mathbf{E}$).

Temporal-Attention Module

The input is a trajectory $\mathbf{E} \in \mathbb{R}^{T\times N\times D}$. To calculate the temporal attention feature, we obtain the transpose of $\mathbf{E}$ as $\bar{\mathbf{E}} \in \mathbb{R}^{N\times T\times D}$. Adopting notation from [88], each row $\mathbf{e}_i \in \mathbb{R}^{T\times D}$ of $\bar{\mathbf{E}}$ is transformed to a *query* $Q_i^{tem} := \mathbf{e}_i W_q^{tem}$, *key* $K_i^{tem} := \mathbf{e}_i W_k^{tem}$, and *value* $V_i^{tem} := \mathbf{e}_i W_v^{tem}$. $W_q^{tem}, W_k^{tem}, W_v^{tem} \in \mathbb{R}^{D\times D}$ are learnable parameters, and $i \in \{0, \ldots, N-1\}$. The t^{th} row $x_{i,t}$ of the temporal attention feature $\mathbf{x}_i \in \mathbb{R}^{T\times D}$ is a weighted sum $x_{i,t} := \alpha_{i,t}^T V_i^{tem}$. The attention weight vector $\alpha_{i,t} \in \mathbb{R}^{T\times 1}$ is a normalization (*softmax*) of the inner-product between the t^{th} row of Q_i^{tem}, $q_{i,t}^{tem}$, and the key matrix K_i^{tem}:

$$\alpha_t^T = \text{softmax}(\frac{q_{i,t}^{tem} K_i^{tem^T}}{\sqrt{D}} \odot m_t^T), \tag{6.1}$$

where $\odot$ is an element-wise product, and m_t is a *mask* with its first t entries equal to 1, and remaining entries 0. The mask preserves causality by ensuring that at any time t, information beyond t will not be used to assign credit. A temporal positional embedding [14]

maintains information about relative positions of states in an episode. Position embeddings are learnable vectors associated to each temporal position of a trajectory. The sum of position and trajectory embeddings forms the input to the temporal attention module. The output of this module is $\mathbf{X} \in \mathbb{R}^{N \times T \times D}$, got by stacking $\mathbf{x}_i, i \in 0, \ldots, N-1$. The temporal attention process can be described by a function $f_{tem}(\mathbf{E}) \rightarrow \mathbf{X}$.

The output of the temporal attention module results in an assessment of each agent's observation at any single time-step relative to observations at other time-steps of an episode. To obtain further insight into how an agent's observation is related to other agents' observations, an agent-attention module is concatenated to the temporal-attention module.

Agent-Attention Module

The agent-attention module uses the transpose of $\mathbf{X}$, denoted $\bar{\mathbf{X}} \in \mathbb{R}^{T \times N \times D}$. Each row of $\bar{\mathbf{X}}$, $\mathbf{x}_t \in \mathbb{R}^{N \times D}$ is transformed to a *query* $Q_t^{agt} = \mathbf{x}_t W_q^{agt}$, *key* $K_t^{agt} = \mathbf{x}_t W_k^{agt}$, and *value* $V_t^{agt} = \mathbf{x}_t W_v^{agt}$. Here, $W_q^{agt}, W_k^{agt}, W_v^{agt} \in \mathbb{R}^{D \times D}$ are learnable parameters. The i^{th} row $z_{t,i}$ of the agent attention feature $\mathbf{z}_t \in \mathbb{R}^{N \times D}$ is a weighted sum, $z_{t,i} = \beta_{t,i}^T V_t^{agt}$. Maintaining causality is not necessary when computing the agent attention weight vector $\beta_{t,i} \in \mathbb{R}^{N \times 1}$. These weights are determined similar to the temporal attention weight vector in Eqn. (6.1), except without a masking operation. Therefore,

$$\beta_{t,i}^T = \text{softmax}(\frac{q_{t,i}^{agt} K_t^{agt^T}}{\sqrt{D}}). \tag{6.2}$$

The agent attention procedure can be described by a function $f_{agt}(\mathbf{X}) \rightarrow \mathbf{Z}$, where $\mathbf{Z} \in \mathbb{R}^{T \times N \times D}$.

Concatenating Attention Modules

The output of the temporal attention module is an entity $\mathbf{X}$ that attends to information at time-steps along the length of an episode for each agent. Passing $\mathbf{X}$ through the agent attention module results in an output $\mathbf{Z}$ that is attended to by embeddings at all time-steps and from all agents. The data-flow of this process can be written as a composition of

functions: $f_{att} := f_{agt} \circ f_{tem}$. The temporal and agent attention modules can be repeatedly composed to improve expressivity. The position embedding is required only at the first temporal attention module when more than one is used.

6.2.3 Credit Assignment

The output of the attention modules is used to assign credit at each time-step along the length of the episode. Let $f_{arel} := f_c \circ (f_{att} \circ \cdots \circ f_{att})$, where $f_c : \mathbb{R}^{T \times N \times D} \to \mathbb{R}^T$. In order to carry out temporal credit assignment effectively, we leverage a property of permutation invariance.

Permutation Invariance

Agents sharing the same action and observation spaces are termed *homogeneous*. When homogeneous agents $ag1$ and $ag2$ cooperate to achieve a goal, the reward when $ag1$ observes $ob1$ and $ag2$ observes $ob2$ should be the same as the case when $ag1$ observes $ob2$ and $ag2$ observes $ob1$. This property is called *permutation invariance*, and has been shown to improve the sample-efficiency of multi-agent credit assignment as the number of agents increase [51, 26]. When this property is satisfied, the output of the function f_{arel} should be invariant to the order of the agents' observations. Formally, if the set of all permutations along the agent dimension (second dimension of $\mathbf{E}$) is denoted $\mathcal{H}$, then $f_{arel}(h_1(\mathbf{E})) = f_{arel}(h_2(\mathbf{E}))$ must be true for all $h_1, h_2 \in \mathcal{H}$.

The function f_{att} is permutation invariant along the agent dimension by design. A sufficient condition for f_{arel} to be permutation invariant is that the function f_c be permutation invariant. To ensure this, we apply a multi-layer perceptron (MLP), add the MLP outputs element-wise, and pass it through another MLP. When functions g_1 and g_2 associated to the MLPs are continuous and shared among agents, the evaluation at time t is the *predicted reward* $\hat{r}_t := g_2\big(\sum_{i=0}^{N-1} g_1(z_{t,i})\big)$. It was shown in [104] that any permutation invariant function can be represented by the above equation.

Remark 3. AREL *can be adapted to the* heterogeneous *case when cooperative agents are divided into homogeneous groups. Similar to a position embedding, we can apply an* agent-group embedding *such that agents within a group share an agent-group embedding. This will maintain permutation invariance of observations within a group, while enabling identification of agents from different groups.* AREL *will also work in the case when the multi-agent system is fully heterogeneous. This is equivalent to a scenario when there is only one agent in each homogeneous group. Therefore,* AREL *can handle agent types ranging from fully homogeneous to fully heterogeneous.*

Credit Assignment Learning

Given a reward R_T at the end of an episode of length T, the goal is to learn a temporal decomposition of R_T to assess contributions of agents at each time-step along the trajectory. Specifically, we want to learn $\{\hat{r}_t\}_{t=0}^{T}$ satisfying $\sum_{t=0}^{T}\hat{r}_t = R_T$. Since $f_{arel}^{\theta}(\mathbf{E})$ is a vector in $\mathbb{R}^T$, its t^{th} entry is denoted $f_{arel}^{\theta}(\mathbf{E}_t)$ $(= \hat{r}_t)$. The sequence $\{\hat{r}_t\}_{t=0}^{T}$ is learned by minimizing a *regression loss*, $l_r(\theta) := \mathbb{E}_{\mathbf{E},\mathbf{R_T}}\big[\frac{1}{T}\big(\sum_t(f_{arel}^{\theta}(\mathbf{E}_t)) - R_T\big)^2\big]$, where θ are neural network parameters.

The redistributed rewards will be provided as an input to a MARL algorithm. We want to discourage $\{\hat{r}_t\}_{t=0}^{T}$ from being sparse, since sparse rewards may impede learning policies [15]. We observe that more than one combination of $\{\hat{r}_t\}_{t=0}^{T}$ can minimize $l_r(\theta)$. We add a regularization loss $l_{reg}(\theta)$ to select among solutions that minimize $l_r(\theta)$. Specifically, we aim to choose a solution that also minimizes the variance $l_v(\theta)$ of the redistributed rewards, and set $l_{reg}(\theta) = l_v(\theta)$ (we examine other choices of $l_{reg}(\theta)$ in the *Appendix*). Using $l_v(\theta)$ as a regularization term in the loss function leads to learning less sparsely redistributed rewards. With $\omega \in \mathbb{R}_{\geq 0}$ denoting a hyperparameter, the combined loss function used to learn f_{arel}^{θ} is:

$$loss_{total}(\theta) = l_r(\theta) + \omega l_v(\theta), \tag{6.3}$$

where $l_v(\theta) := \mathbb{E}_{\mathbf{E}}\big[\frac{1}{T}\sum_t(f_{arel}^{\theta}(\mathbf{E}_t) - \bar{f}^{\theta}(\mathbf{E}))^2\big]$, and $\bar{f}^{\theta}(\mathbf{E}) := \big(\sum_t f_{arel}^{\theta}(\mathbf{E}_t)\big)/T$. The form of $loss_{total}(\theta)$ in Eqn. (6.3) incorporates the possibility that not all intermediate states will

contribute equally to R_T, and additionally results in learning less sparsely redistributed rewards. Note that $\arg\min_\theta[loss_{total}(\theta)]$ will not typically yield $l_r(\theta) = l_v(\theta) = 0$ (which corresponds to a uniform redistribution of rewards). Since some states may be common to different episodes, the redistributed reward $\hat{r}_t$ at each time-step cannot be arbitrarily chosen. For e.g., consider N different episodes $\{E_i\}_{i=1}^N$, each of length L, with distinct cumulative episodic rewards R_i. If an intermediate state s is common to episodes E_j and E_k, under a uniform redistribution, distinct rewards R_j/L and R_k/L will be assigned to s, which is not possible. Thus, $l_r(\theta) = 0$ and $l_v(\theta) = 0$ will not both be true, implying that a uniform redistribution may not be viable.

6.2.4 Algorithm

AREL is summarized in Algorithm 5. Parameters ϕ of RL modules and θ of the credit assignment function are randomly initialized. Observations and actions of agents are collected in each episode (Lines 2-6). Trajectories and episodic rewards are stored in an experience buffer B_e (Line 8). The reward $\hat{r}_t$ at each time step for every trajectory in a batch B_τ (sampled from B_e) is predicted (Lines 9-10). The predicted $\hat{r}_t$ changes as θ is updated, but the episode reward R_T remains the same. A weighted sum $\alpha\hat{r}_t + (1-\alpha)\mathbf{1}_{t=T}R_T$ ($\mathbf{1}_{t=T}$ is an indicator function) is used to update ϕ in a stable manner by using a MARL algorithm (Line 11). The credit assignment function f^θ_{arel} is updated when M new trajectories are available (Lines 12-17).

6.2.5 Analysis

In order to establish a connection between redistributed rewards from *Line 10* of Algorithm 5 and the episodic reward, we define *return equivalence* of *decentralized partially observable sequence-Markov decision processes (Dec-POSDP)*. This generalizes the notion of return equivalence introduced in [2] in the fully observable setting for the single agent case. A Dec-POSDP is a decision process with Markov transition probabilities but has a reward

Algorithm 5 *AREL*

Input: Number of agents N. Reward weight $\alpha \in [0, 1]$.

Initialize parameters θ, ϕ for credit assignment and RL (policy/ critic) modules respectively.

Experience buffer for storing trajectories $B_e \leftarrow \emptyset$. Prediction function update frequency M.

for Episode $k = 0, \ldots$ **do**
  Reset episode return $R_T \leftarrow 0$; Reset trajectory for current episode $\tau \leftarrow \emptyset$
  for step $t = 0, \ldots, T-1$ **do**
    Sample action $a_t^i \sim \pi_\phi^i(o_t^i)$, for $i = 0, \ldots, N-1$
    Take action a_t; Observe $o_{t+1} = (o_{t+1}^0, \ldots, o_{t+1}^{N-1})$
    Store transition $\tau \leftarrow \tau \cup \{(o_t, a_t, o_{t+1})\}$
  end for
  Update episode reward R_T; Store trajectory $B_e \leftarrow B_e \cup (\tau, R_T)$
  Sample a batch of trajectories $B_\tau \sim B_e$
  Predict reward $\hat{r}_t$ using $f_{arel}^\theta(\tau)$ for each $\tau \in B_\tau$
  Update ϕ using $\{(o_t, a_t, o_{o_{t+1}})\} \in \tau$ and weighted reward $\alpha\hat{r}_t + (1-\alpha)\mathbf{1}_{t=T}R_T$
  if k mod M is 0 **then**
    for each gradient update **do**
      Sample a batch from B_e, and compute estimate of total loss, $\hat{loss}_{total}(\theta)$
      $\theta \leftarrow \theta - \nabla_\theta \hat{loss}_{total}(\theta)$
    end for
  end if
end for

distribution that need not be Markov. We present a result that establishes that return equivalent Dec-POSDPs will have the same optimal policies.

Definition 3. *Dec-POSDPs* $\tilde{\mathcal{P}}$ *and* $\mathcal{P}$ *are* ***return-equivalent*** *if they differ only in their*

reward functions but have the same expected return for any trajectory τ.

Theorem 3. *Return-equivalent Dec-POSDPs will have the same optimal policies.*

According to Definition 3, any two return equivalent Dec-POSDPs will have the same expected return for any trajectory τ. That is, $\tilde{R}_0(\tau) = R_0(\tau), \forall \tau$. This is used to prove Theorem 3.

Proof. Consider two return-equivalent Dec-POSDPs $\tilde{\mathcal{P}}$ and $\mathcal{P}$. Since $\tilde{\mathcal{P}}$ and $\mathcal{P}$ have the same transition probability and observation functions, the probabilities that a trajectory τ is realized will be the same if both Dec-POSDPs are provided with the same policy. For any joint agent policy $\pi := (\pi^1, \ldots, \pi^n)$ and sequence of states $s := (s_0, \ldots, s_T)$, we have:

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim(\pi,\tilde{Z},\tilde{P})}\big[\tilde{R}_0(\tau)\big] \\
&\qquad = \sum_{\tau} \tilde{R}_0(\tau) \underbrace{\sum_{s} \tilde{p}(s_0) \prod_{t=0}^{T-1} \pi(a_t|o_t)\tilde{Z}(o_t|s_t)\tilde{p}(s_{t+1}|s_t,a_t)}_{\tilde{p}^{\pi}(\tau)} \\
&\qquad = \sum_{\tau} \tilde{R}_0(\tau) \underbrace{\sum_{s} p(s_0) \prod_{t=0}^{T-1} \pi(a_t|o_t)Z(o_t|s_t)p(s_{t+1}|s_t,a_t)}_{p^{\pi}(\tau)} \\
&\qquad = \sum_{\tau} R_0(\tau) \underbrace{\sum_{s} p(s_0) \prod_{t=0}^{T-1} \pi(a_t|o_t)Z(o_t|s_t)p(s_{t+1}|s_t,a_t)}_{p^{\pi}(\tau)} \\
&\qquad = \mathbb{E}_{\tau\sim(\pi,Z,P)}\big[R_0(\tau)\big].
\end{aligned}
$$

These equations follow from Definition 3. Let π^* denote an optimal policy for $\tilde{\mathcal{P}}$. Then, we have:

$$
\begin{aligned}
\mathbb{E}_{\tau\sim(\pi^*,\tilde{Z},\tilde{P})}\big[\tilde{R}_0(\tau)\big] &= \mathbb{E}_{\tau\sim(\pi^*,Z,P)}\big[R_0(\tau)\big] \\
\geq \mathbb{E}_{\tau\sim(\pi,\tilde{Z},\tilde{P})}\big[\tilde{R}_0(\tau)\big] &= \mathbb{E}_{\tau\sim(\pi,Z,P)}\big[R_0(\tau)\big].
\end{aligned}
$$

Therefore, π^* will also be an optimal policy for $\mathcal{P}$. □

When $l_r(\theta) = 0$ in Eqn. (6.3), a Dec-POSDP with the redistributed reward will be return-equivalent to a Dec-POSDP with the original episodic reward. Theorem 3 indicates that in this scenario, the two Dec-POSDPs will have the same optimal policies.

6.3 Experiments

In this section, we describe the tasks that we evaluate *AREL* on, and present results of our experiments.

6.3.1 Environments and Tasks

We study tasks from *Particle World* [55] and the *StarCraft Multi-Agent Challenge* [74]. These have been identified as challenging multi-agent environments in [55, 74]. In each task, a reward is received by agents only at the end of an episode. No reward is provided at other time steps.

(1) **Cooperative Push**: N agents work together to move a large ball to a landmark.

(2) **Predator-Prey**: N predators seek to capture M preys. L landmarks impede movement of agents.

(3) **Cooperative Navigation**: N agents seek to reach N landmarks. The maximum reward is obtained when there is exactly one agent at each landmark.

(4) **StarCraft**: Units from one group (controlled by RL agents) collaborate to attack units from another (controlled by heuristics). We report results for three maps: 2 Stalkers, 3 Zealots (2s3z); 1 Colossus, 3 Stalkers, 5 Zealots (1c3s5z); 3 Stalkers vs. 5 Zealots (3s_vs_5z).

6.3.2 Architecture and Training

In order to make the agent-temporal attention module more expressive, we use a transformer architecture with multi-head attention [88] for both agent and temporal attention. The permutation invariant critic (*PIC*) based on the multi-agent deep deterministic policy gradient (MADDPG) from [51] is used as the base RL algorithm in Particle World. In StarCraft, we

use QMIX [70] as the base RL algorithm. The value of α is set to 1 in Particle World and 0.8 in StarCraft.

6.3.3 Evaluation

We compare *AREL* with three state-of-the-art methods:
(1) ***RUDDER*** [2]: A long short-term memory (LSTM) network is used for reward decomposition along the length of an episode.
(2) ***Sequence Modeling*** [52]: An attention mechanism is used for temporal decomposition of rewards along an episode.
(3) ***Iterative Relative Credit Refinement (IRCR)*** [26]: 'Guidance rewards' for temporal credit assignment are learned using a surrogate objective.

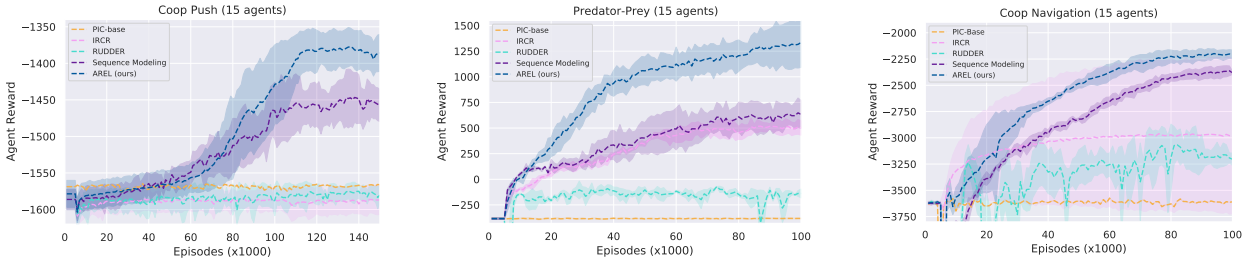
RUDDER and Sequence Modeling were originally developed for the single agent case. We adapted these methods to MARL by concatenating observations from all agents. We added the variance-based regularization loss in our experiments for Sequence Modeling, and observed that incorporating the regularization term resulted in an improved performance compared to without regularization.

6.3.4 Results

AREL *enables improved performance*

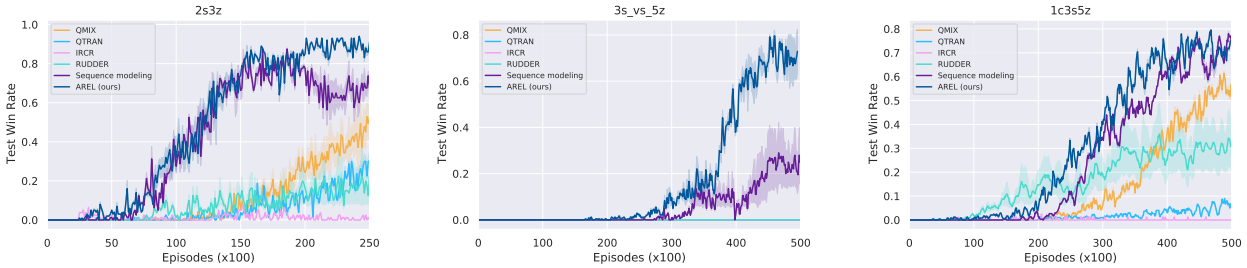
Figure 6.2 shows results of our experiments for tasks in Particle World for $N = 15$. In each case, *AREL* is able to guide agents to learn policies that result in higher average rewards compared to other methods. This is a consequence of using an attention mechanism to redistribute an episodic reward along the length of an episode, and also characterizing the contributions of individual agents.

The *PIC* baseline [51] fails to learn policies to complete tasks with episodic rewards. A similar result of failure to complete tasks was observed when using *RUDDER* [2]. An explanation for this could be that RUDDER only carries out a temporal redistribution of



(a) Cooperative Push (b) Predator and Prey (c) Cooperative Navigation

Figure 6.2: Average agent rewards and standard deviation for tasks in Particle World with episodic rewards and $N = 15$. *AREL* (dark blue) results in the highest average rewards in all tasks.



(a) 2s3z (b) 3s vs 5z (c) 1c3s5z

Figure 6.3: Average test win rate and variance in StarCraft. *AREL* (dark blue) results in the highest win rates in 2s3z and 3s_vs_5z, and obtains a comparable win rate to Sequence Modeling in 1c3s5z.

rewards, but does not consider the effect of agents contributing differently to a reward.

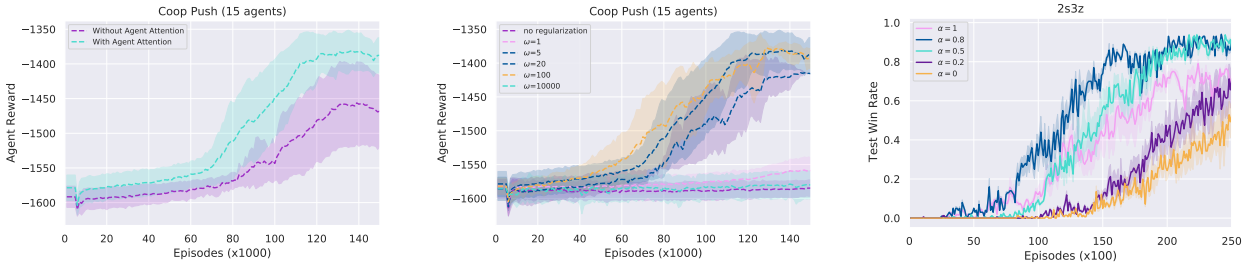
Sequence Modeling [52] performs better than *RUDDER* and the *PIC* baseline, possibly because it uses attention to redistribute episodic rewards. This was shown to outperform LSTM-based models, including *RUDDER*, in [52] in single-agent episodic RL, due to the relative ease of training the attention mechanism. We believe that absence of an explicit characterization of agent-attention resulted in a lower reward for this method compared to *AREL*.

Using a surrogate objective in *IRCR* [26] results in obtaining rewards comparable to *AREL* in some runs in the Cooperative Navigation task. However, the reward when using *IRCR* has a much higher variance compared to that obtained when using *AREL*. A possible reason for this is that *IRCR* does not characterize the relative contributions of agents at intermediate time-steps.

Figure 6.3 shows the results of our experiments for the three maps in StarCraft. *AREL* achieves the highest average win rate in the 2s3z and 3s_vs_5z maps, and obtains a comparable win rate to *Sequence Modeling* in 1c3s5z. *Sequence Modeling* does not explicitly model agent-attention, which could explain the lower average win rates in 2s3z and 3s_vs_5z. *RUDDER* achieves a nonzero, albeit much lower win rate than *AREL* in two maps, possibly because the increased episode length might affect the redistribution of the episode reward for this method. *IRCR* and *QTRAN* [76] obtain the lowest win rates.

Ablation study

We carry out several ablations to evaluate components of *AREL*. Figure 6.4a demonstrates the impact of the agent-attention module. In the absence of agent-attention (while retaining permutation invariance among agents through the shared temporal attention module), rewards are significantly lower. We study the effect of the value of ω in Eqn. (6.3) on rewards in Figure 6.4b. This term is critical in ensuring that agents learn good policies. This is underscored by observations that rewards are significantly lower for very small or very large ω ($\omega = 0, \omega = 1, \omega = 10000$). Third, we evaluate the effect of mixing the original episodic



(a) Agent Attention (b) Regularization (c) Reward Weight

Figure 6.4: Ablations: Effects of the agent attention module (Fig. 6.4a) and regularization parameter ω in Eqn (6.3) (Fig. 6.4b) in Cooperative Push, and reward weight α (Fig. 6.4c) in the 2s3z StarCraft map.

reward and redistributed reward by changing the reward weight α in Figure 6.4c. The reward mixture influences win rates; $\alpha = 0.5$ or 0.8 yields the highest win rate. The win rate is $\sim 10\%$ lower when using redistributed reward alone ($\alpha = 1$).

Credit Assignment vs. exploration

This section demonstrates the importance of effective redistribution of an episodic reward *vis-a-vis* strategic exploration of the environment. The episodic reward $R_T(= \sum_t r_t)$ takes continuous values and provides fine-grained information on performance (beyond only win/ loss). AREL learns a redistribution of R_T by identifying critical states in an episode, and does not provide exploration abilities beyond that of the base RL algorithm. The redistributed rewards of AREL can be given as input to any RL algorithm to learn policies (in our experiments, we demonstrate using QMIX for StarCraft; MADDPG for Particle World). Figure 6.5 illustrates a comparison of *AREL* with a state-of-the-art exploration strategy, MAVEN [56] and with QMIX [70] in the $3s\_vs\_5z$ StarCraft map. We observe that when rewards are delayed to the end of an episode, effectively redistributing the reward can be more beneficial than strategically exploring the environment to improve win-rates or total

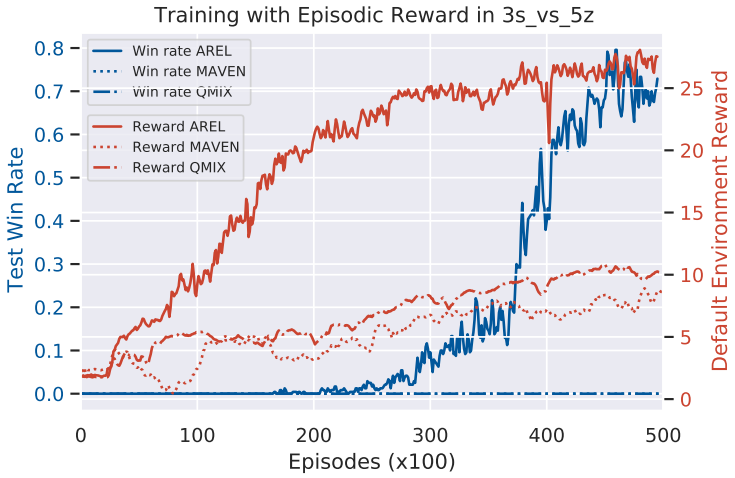


Figure 6.5: Comparison of AREL with QMIX and a strategic exploration technique, MAVEN in the $3s\_vs\_5z$ StarCraft map (avg. over 5 runs). AREL yields highest rewards and win rates.

rewards.

Interpretability of Learned Rewards

Figure 6.6 presents an interpretation of the decomposed predicted rewards *vis-a-vis* the relative positions of agents to landmarks in the Cooperative Navigation task with $N = 3$. When the reward is provided only at the end of an episode, *AREL* is used to learn a temporal redistribution of this episodic reward. The predicted rewards are normalized to a $0 - 1$ scale for ease of representation. The positions of the agents relative to the landmarks are shown at several points along a sample trajectory. Successfully trained agents must learn policies that enable each agent to cover a distinct landmark. For example, in a scenario where two agents are close to a single landmark, one of them must remain close to this landmark, while the other moves towards a different landmark. We observe that the magnitude of the predicted rewards is consistent with this insight in that it is higher when agents navigate away and towards different landmarks.

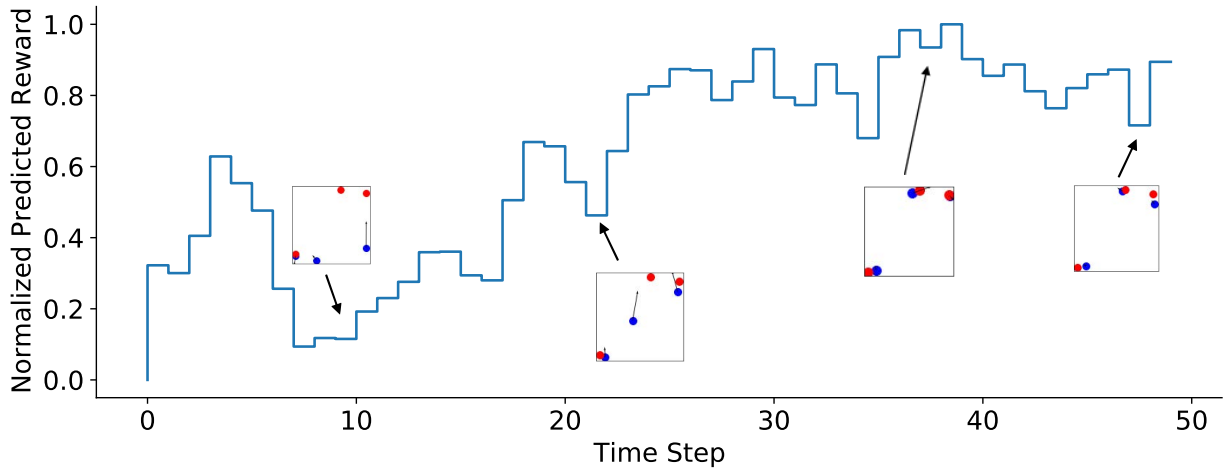


Figure 6.6: An instantiation of the Cooperative Navigation task with $N = 3$ where rewards are provided only at the end of an episode. Blue and red dots respectively denote agents and landmarks. Arrows on agents represent their directions of movement. The objective of this task is for each agent to cover a distinct landmark. *AREL* is used to learn a redistribution of the episodic reward along the length of the episode. The $y-$axis of the graph shows the $0-1$ normalized predicted rewards for a sample trajectory. The positions of agents relative to landmarks are shown at several points along this trajectory. The figure shows a scenario where two agents are close to a single landmark. In this case, one of them must remain close to this landmark, while the other moves towards a different landmark. The predicted redistributed reward encourages such an action, since it has a higher magnitude when agents navigate towards distinct landmarks. The predicted redistributed reward by *AREL* is not uniform along the length of the episode.

This visualization in Figure 6.6 reveals that the attention mechanism in *AREL* is able to learn to redistribute an episodic reward effectively in order to successfully train agents to accomplish task objectives in cooperative multi-agent reinforcement learning. Moreover, it reveals that the redistributed reward predicted by *AREL* is not uniform along the length of the episode.

6.3.5 Discussion

This chapter focused on developing techniques to effectively learn policies in MARL environments when rewards were episodic. Our experiments demonstrate that *AREL* can be used as a module that enables more effective credit assignment by identifying critical states through capturing long-term temporal dependencies between states and an episodic reward. Redistributed rewards predicted by *AREL* are dense, which can then be provided as an input to MARL algorithms that learn value functions for credit assignment (we used MADDPG [55] and QMIX [70] in our experiments).

By including a variance-based regularization term, the total loss in Eqn. (6.3) enabled incorporating the possibility that not all intermediate states would contribute equally to an episodic reward, while also learning less sparse redistributed rewards. Moreover, any exploration ability available to the agents was provided solely by the MARL algorithm, and not by *AREL*. We further demonstrated that effective credit assignment was more beneficial than strategic exploration of the environment when rewards are episodic.

6.4 Summary

This chapter studied the multi-agent temporal credit assignment problem in MARL tasks with episodic rewards. Solving this problem required addressing the twin challenges of identifying the relative importance of states along the length of an episode and individual agent's state at any single time-step. We presented an attention-based method called *AREL* to deal with the above challenges in a data-driven way. The temporally redistributed reward predicted by *AREL* was dense, and could be integrated with MARL algorithms. *AREL* was evaluated on tasks from Particle World and StarCraft, and was compared with three state-of-the-art reward redistribution techniques. *AREL* resulted in agents obtaining higher rewards in Particle World and better win rates in StarCraft.

Chapter 7

CONCLUSIONS

The research contributions and experimental results in this thesis have demonstrated that reward shaping is an effective way to improve the performance of reinforcement learning in a variety of situations where the environment reward signal is sparse or delayed. For both single-agent and multi-agent reinforcement learning, we have developed methods to effectively incorporate potential-based advice. In Chapter 3, this thesis reported the use of PBRS and PBA for learning stochastic policies in single agent systems. We then explored how to make use of potential-based advice in multi-agent learning systems in Chapter 4, where the *SAM* architecture is presented. In the absence of potential-based advice, we investigated the possibilities of putting a human in the learning loop to provide shaping reward, and the *FRESH* framework is proposed in Chapter 5. FRESH was evaluated on the *Bowling* and *Skiing* Atari games of the arcade learning environment and showed much better performance than state-of-the-art deep learning algorithms. For scenarios where both potential-based advice and human feedback were not available, we explored data-driven possibilities in Chapter 6. We studied the multi-agent temporal credit assignment problem in MARL tasks with episodic rewards. Solving this problem required addressing the twin challenges of identifying the relative importance of states along the length of an episode and individual agent's state at any single time-step. We presented an attention-based method called *AREL* to deal with the above challenges. *AREL* was evaluated on tasks from Particle World and StarCraft, and was compared with three state-of-the-art reward redistribution techniques. *AREL* resulted in agents obtaining higher rewards in Particle World and better win rates in StarCraft.

There are several future directions that could be explored. The potential-based methods

investigated in this thesis assume that potential functions are available for reward shaping. However, potential functions may not be available, especially when the environment is complex. Without prior domain knowledge, we could explore how to learn a potential function at the same time with reinforcement learning, such that the learned potential function can in turn improve the reinforcement learning procedure. AREL only decomposes the episodic global reward along temporal steps, which means different agents receive the same reward at each time-step. In order to speed up the learning process, we could investigate methods to decompose the reward along the agent dimension so that different agents will receive different rewards even at the same time-step. The results presented in this thesis will form a foundation to develop techniques to address the above challenges, which will allow the realization of reward shaping methods in real-world applications.

BIBLIOGRAPHY

[1] Riku Arakawa, Sosuke Kobayashi, Yuya Unno, Yuta Tsuboi, and Shin-ichi Maeda. DQN-TAMER: Human-in-the-loop reinforcement learning with intractable feedback. *arXiv preprint arXiv:1810.11748*, 2018.

[2] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, pages 13566–13577, 2019.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11), 2009.

[5] Reinaldo AC Bianchi, Murilo F Martins, Carlos HC Ribeiro, and Anna HR Costa. Heuristically-accelerated multiagent reinforcement learning. *IEEE Transactions on Cybernetics*, 44(2):252–265, 2013.

[6] Vivek S Borkar. *Stochastic approximation: A dynamical systems viewpoint*, volume 48. Springer, 2009.

[7] V.S. Borkar and S.P. Meyn. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2), 2000.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai Gym. *arXiv:1606.01540*, 2016.

[9] Daniel S Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. *arXiv preprint arXiv:1904.06387*, 2019.

[10] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[11] Hyeong Soo Chang. Reinforcement learning with supervision by combining multiple learnings and expert advices. In *Proceedings of the American Control Conference*, pages 4159–4164, 2006.

[12] Yu-Han Chang, Tracey Ho, and Leslie P Kaelbling. All learning is local: Multi-agent learning in global reward games. In *Neural Information Processing Systems*, pages 807–814, 2004.

[13] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 225–232, 2011.

[16] Sam Devlin, Daniel Kudenko, and Marek Grześ. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(02):251–278, 2011.

[17] Sam Michael Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Autonomous Agents and Multiagent Systems*, pages 433–440, 2012.

[18] Gaurav Dixit, Stéphane Airiau, and Kagan Tumer. Gaussian processes as multiagent reward models. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 330–338, 2020.

[19] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In *Neural Information Processing Systems*, pages 4405–4416, 2019.

[20] Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994.

[21] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1406–1415, 2018.

[22] Taylor Kessler Faulkner, Elaine Schaertl Short, and Andrea Lockerd Thomaz. Policy shaping with supervisory attention driven exploration. In *International Conference on Intelligent Robots and Systems*, pages 842–847. IEEE, 2018.

[23] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1146–1155, 2017.

[24] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, pages 2974–2982, 2018.

[25] Yarin Gal and Zoubin Ghahramani. Dropout as Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 2016.

[26] Tanmay Gangwani, Yuan Zhou, and Jian Peng. Learning guidance rewards with trajectory-space smoothing. In *Neural Information Processing Systems*, 2020.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[28] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2625–2633, 2013.

[29] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83, 2017.

[30] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies. In *International Conference on Machine Learning*, pages 1352–1361, 2017.

[31] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv:1801.01290*, 2018.

[32] Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84:137–169, 2011.

[33] Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowé. Expressing arbitrary reward functions as potential-based advice. In *AAAI*, pages 2652–2658, 2015.

[34] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[35] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, pages 3223–3230, 2018.

[36] Charles Isbell, Christian R Shelton, Michael Kearns, Satinder Singh, and Peter Stone. A social reinforcement learning agent. In *Proceedings of the International Conference on Autonomous Agents*, pages 377–384, 2001.

[37] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2019.

[38] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *International Conference on Robotics and Automation*, pages 8077–8083. IEEE, 2019.

[39] Taylor Kessler Faulkner, Reymundo A Gutierrez, Elaine Schaertl Short, Guy Hoffman, and Andrea L Thomaz. Active attention-modified policy shaping: Socially interactive agents track. In *Autonomous Agents and MultiAgent Systems*, pages 728–736, 2019.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[41] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *International Conference on Knowledge Capture*, pages 9–16, 2009.

[42] W Bradley Knox and Peter Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Autonomous Agents and Multiagent Systems*, pages 5–12, 2010.

[43] W Bradley Knox and Peter Stone. Reinforcement learning from simultaneous human and mdp reward. In *Autonomous Agents and Multiagent Systems*, pages 475–482, 2012.

[44] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014, 2000.

[45] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[46] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 464–473, 2017.

[47] 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.

[48] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[49] Mao Li, Tim Brys, and Daniel Kudenko. Introspective reinforcement learning and learning from demonstration. In *Autonomous Agents and MultiAgent Systems*, pages 1992–1994, 2018.

[50] Timothy P Lillicrap et al. Continuous control with deep reinforcement learning. In *International Conference on Learning and Representations*, 2016.

[51] Iou-Jen Liu, Raymond A Yeh, and Alexander G Schwing. Pic: Permutation invariant critic for multi-agent deep reinforcement learning. In *Conference on Robot Learning*, pages 590–602, 2020.

[52] Yang Liu, Yunan Luo, Yuanyi Zhong, Xi Chen, Qiang Liu, and Jian Peng. Sequence modeling of temporal credit assignment for episodic reinforcement learning. *arXiv preprint arXiv:1905.13420*, 2019.

[53] Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. Multi-agent game abstraction via graph attention neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7211–7218, 2020.

[54] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. Learning behaviors via human-delivered discrete feedback: Modeling implicit feedback strategies to speed up learning. *Autonomous agents and multi-agent systems*, 30(1):30–59, 2016.

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

[56] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: Multi-agent variational exploration. *arXiv preprint arXiv:1910.07483*, 2019.

[57] Patrick Mannion, Sam Devlin, Jim Duggan, and Enda Howley. Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *The Knowledge Engineering Review*, 33, 2018.

[58] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, and Zhibo Gong. Modelling the dynamic joint policy of teammates with attention multi-agent ddpg. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1108–1116, 2019.

[59] Ofir Marom and Benjamin Rosman. Belief reward shaping in reinforcement learning. In *AAAI*, pages 3762–3769, 2018.

[60] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

[61] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015.

[62] Volodymyr Mnih et al. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[63] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[64] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[65] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[66] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, 2016.

[67] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, 2017.

[68] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[69] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *International Conference on Machine Learning*, 1998.

[70] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304, 2018.

[71] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[72] Heechang Ryu, Hayong Shin, and Jinkyoo Park. Multi-agent actor-critic with hierarchical graph attention network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7236–7243, 2020.

[73] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[74] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[75] David Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 2016.

[76] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896, 2019.

[77] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z

Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 2085–2087, 2018.

[78] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[79] 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.

[80] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS One*, 12(4):e0172395, 2017.

[81] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning*, pages 330–337, 1993.

[82] Haoran Tang et al. # Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

[83] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 617–624, 2011.

[84] Andrea Lockerd Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI*, pages 1000–1005, 2006.

[85] Zheng Tian, Ying Wen, Zhichen Gong, Faiz Punakkath, Shihao Zou, and Jun Wang. A regularized opponent model with maximum entropy objective. In *International Joint Conference on Artificial Intelligence*, pages 602–608, 2019.

[86] Kagan Tumer and Adrian Agogino. Distributed agent-based air traffic flow management. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 1–8, 2007.

[87] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, pages 5998–6008, 2017.

[89] Zhaodong Wang and Matthew E Taylor. Improving reinforcement learning with confidence-based demonstrations. In *International Joint Conference on Artificial Intelligence*, pages 3027–3033, 2017.

[90] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[91] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep TAMER: Interactive agent shaping in high-dimensional state spaces. In *AAAI Conference on Artificial Intelligence*, pages 1545–1554, 2018.

[92] Ying Wen, Yaodong Yang, Rui Luo, Jun Wang, and W Pan. Probabilistic recursive reasoning for multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2019.

[93] Eric Wiewiora. Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, pages 205–208, 2003.

[94] Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *International Conference on Machine Learning*, pages 792–799, 2003.

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

[96] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *The Journal of Machine Learning Research*, 18(1):4945–4990, 2017.

[97] 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.

[98] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827, 2019.

[99] Baicen Xiao, Qifan Lu, Bhaskar Ramasubramanian, Andrew Clark, Linda Bushnell, and Radha Poovendran. FRESH: Interactive reward shaping in high-dimensional state spaces using human feedback. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1512–1520, 2020.

[100] Baicen Xiao, Bhaskar Ramasubramanian, Andrew Clark, Hannaneh Hajishirzi, Linda Bushnell, and Radha Poovendran. Potential-based advice for stochastic policy learning. In *IEEE Conference on Decision and Control*, pages 1842–1849. IEEE, 2019.

[101] Baicen Xiao, Bhaskar Ramasubramanian, and Radha Poovendran. SAM: Advising agents for credit assignment in deep multi-agent reinforcement learning. *Under Review*.

[102] Yaodong Yang, Jianye Hao, Guangyong Chen, Hongyao Tang, Yingfeng Chen, Yujing Hu, Changjie Fan, and Zhongyu Wei. Q-value path decomposition for deep multiagent reinforcement learning. In *International Conference on Machine Learning*, 2020.

[103] Z. Yang, K. Zhang, M. Hong, and T. Başar. A finite sample analysis of the actor-critic algorithm. In *IEEE Conference on Decision and Control (CDC)*, pages 2759–2764, 2018.

[104] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. In *Neural Information Processing Systems*, 2017.

[105] Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H Ballard, and Peter Stone. Leveraging human guidance for deep reinforcement learning tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 6339–6346, 2019.

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