Efficient Game Solving Through Transfer Learning

by

Max Olan Smith

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2023

Max Olan Smith

mxsmith@umich.edu

ORCID iD: 0000-0003-1783-8937

ACKNOWLEDGMENTS

Research is not a solitary pursuit, as each manuscript contributes to the collective discourse of human knowledge. Falling along these lines, the work contained herein, would not have been possible without a large number of people. I would first like to thank my advisor Michael P. Wellman, whose kindness and insights were paramount to this dissertation. I joined Michael's lab without any experience in Game Theory, and thanks to his patience and mentorship, I was able to develop this uniquely interdisciplinary work. I would also like to thank the remainder of my doctoral committee for their constructive feedback on my work throughout my degree: Satinder Singh, Honglak Lee, and Grant Schoenebeck. I am also particularly grateful for the opportunity to collaborate with Thomas Anthony, who played an instrumental role in the development of this dissertation. I am also eternally grateful for the mentorship of John Laird, without his advice, this work would not have occurred.

I next thank my former and current Michigan colleagues for their time and attention: Megan Shearer, Katherine Mayo, Madelyn Gatchel, Yongzhao Wang, Austin Nguyn, Christine Konicki, Zun Li, Xintong Wang, Mithun Chakraborty, Junhyuk Oh, Christopher Grimm, Qi Zhang, Zeyu Zheng, Vivek Veeriah, Ethan Brooks, Wilka Carvalho, Janarthanan Rajendran, Risto Vuorio, Aditya Modi, Shun Zhang, Jonathan Kummerfeld, Mahmoud Azab, Mingzhe Wang, Andrew DeOrio, Rada Mihalcea, Jia Deng, Changhan Wang, and Xinchen Yan. I am also grateful for the opportunity to visit the DeepMind Paris team during my doctoral work. This presented the unique opportunity to learn from: Daniel Hennes, Paul Muller, Romuald Elie, Eugene Tarassov, Julien Perolat, Michael Kaisers, Karl Tuyls, Florian Strub, John Schultz, Neil Burch, and Marc Lanctot.

Beyond work, I am also indebted to many people for making my time so memorable. I owe much of who I am today to my experiences with my housemates in Prospect: Richard Higgins, Alex "Bubbles" Kapiamba, Timothy Trippel, Jonathan Stroud, Alejandro Newell, Mohamed El Banani, Matthew Perez, Lauren Willigan, Ian Fox, and Yan Rou Yap. I am also thankful to Barrett Olson, Katelyn Boisvert, Melanie Schroeder, Karan Desai, Barbara Korycki, James Boggs, Amelia Grant-Alfieri, Oskar Singer, and many more for making Ann Arbor feel like home. I also appreciate the emotional support of Claudia, Roo, and Lulu the goodest of girls. Lastly, I express my deepest gratitude to my family for their unwavering love and support throughout this journey.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

AI Artificial Intelligence

A3C Asynchronous Advantage Actor-Critic

ABR Approximate Best Response

BR Best Response

DL Deep Learning

DO Double Oracle

DNN Deep Neural Network

DQN Deep Q-Network

DRL Deep Reinforcement Learning

EGTA Empirical Game Theoretic Analysis

ENFG Empirical Normal-Form Game

GT Game Theory

IS Importance Sampling

IMPALA Importance Weighted Actor-Learner Architecture

MARL Multiagent Reinforcement Learning

MAS Multiagent System

MBRL Model-Based Reinforcement Learning

MSS Meta-Strategy Solver

NE Nash Equilibrium

NFG Normal-Form Game

OLM Opponent Likelihood Model

OPC Opponent Policy Classifier

PSRO Policy-Space Response Oracles

RL Reinforcement Learning

RGB Red Green Blue

RPS Rock Paper Scissors

RWS Running With Scissors

LIST OF SYMBOLS

s	State or information state.
$\mathcal{S}$	State or information state space.
o	Observation.
$\mathcal{O}$	Observation space.
h	Game state.
$\mathcal{H}$	Game state space.
a	Action.
$\mathcal{A}$	Action space, that is optionally conditioned on a state $\mathcal{A}(s)$.
r	Reward.
$\mathcal{R}$	Reward space.
γ	Discount Factor $[0, 1]$.
τ	Trajectory (sequence) of experiences.
n	Number of players in a game.
i	Player index, where $-i$ refers to all but player i.
Γ	Game.
$\hat{\Gamma}$	Empirical game, with game components similarly denoted with a hat.
σ	Strategy defining a player's selection of policy at the beginning of a game.
π	Policy defining a player's behavior.
Π	Space of policies, also defining the full strategy-set of an agent.
V	Value function.
Q	Action-value function.
w	Agent world model, representing the dynamics in terms of information available to the agent.
θ	Model parameters, for example the weights of a neural network.

ABSTRACT

Game-solving approaches using reinforcement learning often entail a significant computational cost. This arises from the necessity of training agents to play with or against a series of other-agent strategies. Each round of training brings us closer to the game's solution, but training an agent can require data from millions of games played—typically in simulation. The cost of game solving reflects the cumulative data cost of repeatedly training agents. This cost is also a result of treating each training as an independent problem. However, these problems share elements that reflect the nature of the game-solving process. These similarities present an opportunity for an agent to transfer learning from previous problems to aid in solving the current problem.

In this dissertation, I develop a collection of new game-solving algorithms that are based on new methods for transfer learning, thereby reducing the computational cost of game solving. I explore two types of transferable knowledge: strategic and world. Strategic knowledge describes knowledge that depends on the other agents. In the simplest case, strategic knowledge may be encapsulated in a policy that was trained to play, with or against, fixed other agents. To facilitate the transfer of this kind of strategic knowledge, I propose Q-Mixing, a technique that constructs a policy to play against a distribution of other agents by combining strategic knowledge regarding each agent in the distribution. I provide a practical approximate version of Q-Mixing that features another type of strategic knowledge: a learned belief in the distribution of the other agents. I then develop two game-solving algorithms, Mixed-Oracles and Mixed-Opponents. These algorithms use Q-Mixing to shift the learning focus from interacting with a distribution of other agents to concentrating on a single other agent. This transition results in a significantly easier and, therefore, less costly learning problem. Complementary to strategic knowledge, world knowledge is independent of the other agents. I demonstrate that co-learning a world model along with game solving allows the world model to benefit from more strategically diverse training data. It also renders game solving more affordable through planning. I realize both of these benefits in a new game-solving algorithm Dyna-PSRO. Overall, this dissertation introduces new techniques and demonstrates their effectiveness in significantly reducing the cost of game solving. By doing so, it further enables learning-based game-solving algorithms to be applied to more complex games.

Part I

Preliminaries

CHAPTER 1

Introduction

Consider a trip to your local farmer's market for weekly grocery shopping. Every Saturday morning, this journey sees you gathering fresh produce from a varying lineup of stalls, with staples like onions, potatoes, and bell peppers typically making up your shopping list. However, your meal plan remains flexible, always on the lookout for the freshest, highest-quality seasonal goods. While this routine might appear straightforward at first glance, it underscores the complexities inherent in even the most everyday human tasks. Indeed, this market scenario is a *game*, and our ability to efficiently navigate it highlights an innate ability to draw upon a lifetime of previous experiences playing it.

Now, envision the same excursion to the market, but with the need to *relearn* its mechanics at every visit. You find yourself continually needing to familiarize yourself with its explicit and often tacit rules. What is the layout of the vendors? Are their locations fixed, or do you need to continually search for your favorite baker? What payment methods are accepted? These questions barely scratch the surface of the market's basic operations, let alone the additional complexities introduced by its participants. Securing limited, high-demand goods might necessitate arriving at specific times. Vendors might be open to bargaining, offering discounts for rapport or bundled items. And even excessive queues may require replanning your trip.

However, as humans, we are not hampered by the ongoing need to relearn these dynamics with each market visit. This resilience can be credited to our ability to *transfer* knowledge. This essential skill allows us to apply lessons learned from a host of related tasks throughout our lives to swiftly navigate new circumstances. Naturally, regular market visits enhance our understanding with its processes, thereby increasing our efficiency. Over time changes in the market may occur that require a change in our approach. These changes can be due to rotating vendors, or the seasonality of goods. Regardless, we only need to learn about the changes, transferring our knowledge of familiar aspects of the problem.

To illustrate the power of knowledge transfer, let's consider a situation where your goal is to buy a bâtard, an oval-shaped loaf of bread. You direct your steps towards the usual spot of White Lotus Farms, famous for their artisanal breads and pastries. Unexpectedly, you find the stall occupied by Detroit Mushroom Company, with White Lotus Farms nowhere to be seen that week. While this shift offers an unexpected chance to buy some seasonal morels, it interrupts your plan to acquire a bâtard. However, during your market exploration, you noticed a stall for Rye Humor Baking. Despite being a New York-based baker, they are familiar to you from their stint in your town years ago. From them, you manage to secure a multiseed rye boule, an enticing substitute for the bâtard. Opting to pay in cash, you evade credit card fees for both yourself and the small business, although it's slightly less convenient—a lesson learned from a past experience.

This example demonstrates how the transference of a lifetime of experiences can assist in addressing new challenges. The transferred knowledge can encompass everything from comprehending the market's functioning to perceiving the behaviors of other market-goers. In this dissertation, I delve into this concept further by interpreting the series of learning challenges in learning-based game-solving algorithms as a transfer learning problem. I develop algorithms that train artificially intelligent (AI) agents to effectively play games by combining principles from both reinforcement learning (RL) and game theory (GT). Through this perspective, I outline a classification of transferable knowledge and propose a suite of algorithms to facilitate their transfer. I subsequently demonstrate the capacity of these algorithms to considerably lessen the cost of game solving.

1.1 Learning Solutions to Games

What defines successful behavior for an agent within a multiagent system? To understand this, let's first explore the context of a single-agent system. In this setting, the system itself establishes the agent's goal. Success is measured by progress towards the completion of this goal. After every decision, the system provides feedback to the agent through a reward signal. The aggregate reward garnered by the agent, termed as the return, serves as a straightforward metric for success.

Measuring performance becomes more nuanced when transitioning from single-agent to multiagent systems. This complexity arises because an agent's reward is dependent not just on its own decisions, but also on those made by all other agents within the system. I hereafter refer to "all other agents" as coplayers or opponents. To facilitate this measure, we could assume certain behaviors of our coplayers. For instance, we might posit that our coplayers are rational and consistently strive to maximize their individual rewards. This act of making assumptions essentially defines a solution concept—a formal rule that predicts the behaviors exhibited by players in a game. Once we account for our coplayers through the selection of a solution concept, return appears to be a fitting measure of success.

However, using return against a solution concept introduces a cause-effect dilemma. This measurement requires us to have already solved the game, because we need the solution-playing coplayers to evaluate against. In small or toy games, this is not a problem, because the game can be analytically described, facilitating direct computation of a solution. Even so, computing a solution quickly becomes infeasible as the complexity of the game increases even in ostensibly simple games.

Up to this point, we have pinpointed two essential requirements for a general method of training an agent in a game. First, the method should train an agent to maximize their return against a chosen solution concept. Second, the method should predict the behaviors for all players. These aspects encapsulate the vital relationship between learning effective behavior and discovering a game's solution.

The need to fulfill both desiderata has spurred the development of techniques in Empirical Game Theoretic Analysis (EGTA) [Wellman, 2006, Tuyls et al., 2020]. EGTA methods construct an approximate model of a game termed the *empirical game* that is used as a proxy for the true game of interest. A more precise definition is provided in Chapter 2, but for now, think of an empirical game as an approximate subgame—a game containing a subset of the possible behaviors. An illustration of an empirical game is included in Figure 1.1, which depicts a complex world by a simple payoff matrix. Due to the diminished size of the empirical game, it affords analytic solving and reasoning. The outcomes from this process can be utilized as approximations for the equivalent quantities in the actual game.

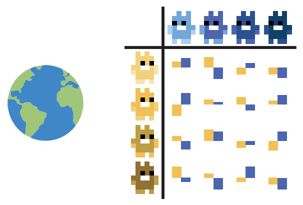


Figure 1.1: **Empirical game conceptual example.** The possible space of behaviors for a player is every shade of their respective color. The empirical game captures only a sample of the possible behaviors. Payoff estimates are bar graphs for each combination of behaviors.

Before we can employ an empirical game, we must first address how to construct one. The construction process unfolds by alternating between two subroutines: game reasoning and strategy exploration. Game reasoning involves operations that analyze the current state of the empirical game. Common game-reasoning procedures include determining its current solution and refining estimated payoffs, as examplified in Figure 1.2. Strategy exploration expands the quality of the empirical game through the inclusion of additional behaviors. This raises a key question in empirical-game modeling: how to select which behaviors to incorporate into the model. This general question has been termed the *strategy exploration problem* by Jordan et al. [2010], and plays a critical role in the quality of an empirical game, and as we will see later, the cost of its construction.

Repeated execution of strategy exploration enlarges the empirical game, as shown in Figure 1.3. After each expansion, game reasoning is typically performed to inform subsequent expansions. Moreover, game reasoning serves the dual purpose of checking if the empirical game's solution has converged.

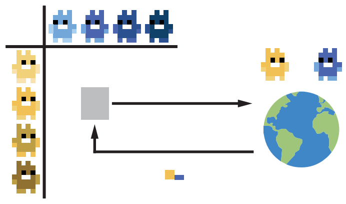


Figure 1.2: **Profile payoff estimation.** A profile payoff, gray box, is estimated by playing the corresponding policies for many games and averaging the player's payoffs.

To facilitate our discussion of strategy exploration, we need to first refine our terminology about behavior. A *policy* is a complete description of an agent's behavior in every possible state of the game. Therefore, the strategy exploration problem seeks to determine which policy should be next included in the empirical game.

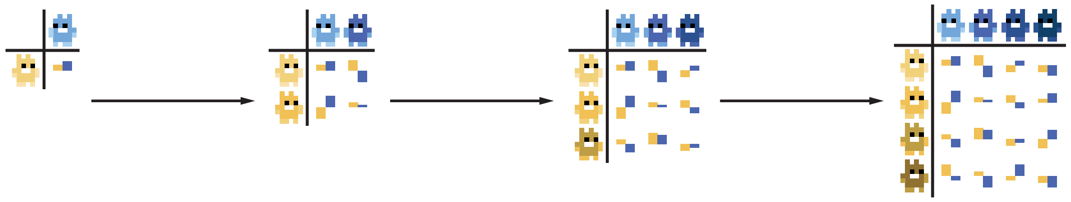


Figure 1.3: **Strategy exploration iteratively expanding an empirical game.** Example of repeated applications of strategy exploration, the arrow, expanding an empirical game.

A performant answer to this question is to include the best-response (BR) policy to the empirical game's current solution [Schvartzman and Wellman, 2009a]. This method has demonstrated empirical success when the chosen solution concept is a Nash equilibrium (NE) [Nash Jr, 1950]—a prediction that all players will maximize their own return. However, despite its success, computing exact best responses is often impracticable due to the complexity of the problem, which rivals the direct computation of the game's solution. This complexity has necessitated methods that can effectively compute approximate best-response (ABR) policies, also referred to as response policies.

RL is a logical choice for a method for computing response policies. Schvartzman and Wellman [2009b] initially demonstrated the efficacy of RL as an ABR method for EGTA. RL computes an ABR by training an agent to maximize its return to any fixed configuration of coplayers. In this approach, RL computes an ABR by training an agent to maximize its return against a fixed configuration of coplayers. In the context of game solving, this fixed configuration corresponds to the coplayers adhering to the solution of the current empirical game.

Policy-Space Response Oracles (PSRO) is one such realization of a RL-based EGTA algorithm [Lanctot et al., 2017]. PSRO expanded the space of considered ABR computation methods to include deep learning (DL) based RL methods, so-called deep reinforcement learning (DRL) methods. The incorporation of DL allows EGTA to be applied to games with complex state information, such as high-resolution images, an obstacle that traditional, non-DRL methods were computationally impractical.

In summary, it's evident that learning to play a game is not a straightforward undertaking. It requires not only mastering an effective policy for a single player, but also learning policies for all players that conform to the intended solution concept. However, due to the inherent complexity present even in seemingly small games, directly solving these problems is infeasible. As such, rather than solving the game directly, an empirical game is constructed and solved as a surrogate. The construction of an empirical game necessitates the repeated application of DRL to compute new policies for inclusion in the empirical game.

1.2 Transfer Learning in Game Solving

The application of DRL often entails significant costs. These costs stem from the method's need for extensive game-playing experiences to train a policy—a complete description of a behavior. In real-world scenarios, each gameplay session can demand considerable time with human participants [Hester and Stone, 2012]. To circumvent the need for human interactions or other real-world expenses, we often turn towards computer simulations.

Despite these measures, DRL can still necessitate billions of experiences to effectively train a policy [Obando-Ceron and Castro, 2021]. Moreover, even when these experiences are simulated using a distributed high-performance computing center, they can still demand a considerable amount of time, ranging from several days to months, to generate [Vinyals et al., 2019]. These costs serve as a limiting factor for the practicality of DRL, consequently restricting the utilization of PSRO due to the need for repeated applications of DRL.

Nonetheless, there is reason for optimism. The high computational cost of PSRO is a result of its default treatment of each ABR calculation as an independent problem. This treatment overlooks that these problems are intrinsically interconnected through a shared structure that reflects their respective step in the empirical game solving process. To reveal the nature of this structure, we can examine their common components.

Figure 1.4 illustrates consecutive ABR problems encountered in a run of PSRO. Firstly, consider the game itself. Given that the formulation of an empirical game is aimed at solving a singular real game, all ABR computations must involve the same game. Concurrently, the strategy sets of the empirical game expand incrementally with each ABR calculation, marked by the inclusion of policies obtained from the preceding iteration (in this example depicted, one dark-blue policy). The distribution that coplayers utilize to choose their policy generally evolves, mirroring insights gleaned from the interim game-reasoning step. These common structures offer abundant opportunities for transfer learning. Efficient transfer learning will allow us to focus exclusively on the novel aspects of the problem, thereby enabling us to optimize the use of our costly experiences.

In fact, the series of ABR problems proposed by PSRO resembles a subproblem in transfer learning called lifelong learning [Thrun, 1995]. The lifelong learning framework proposes a scenario where a learner is confronted with a sequence of tasks, using knowledge gained from previous tasks to assist with the current task. In the context of PSRO, each task is represented by an ABR calculation, and the "lifetime" of the learner corresponds to the game-solving process. Drawing an analogy, an agent might spend its entire lifetime learning to play the game of life. However, unlike life, the empirical game-solving process introduces a shared and predictable structure into the sequence of tasks. The pressing question becomes: what should we transfer, and how should we transfer it across ABR calculations?

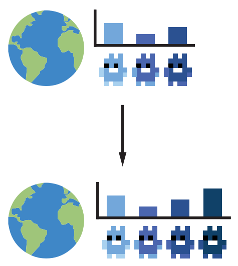


Figure 1.4: **Best-Response Problem Comparison.** Best response problems for the yellow player when building an empirical game, such as in Figure 1.1. The blue player's strategy set only marginally changes. The bar graph about the opponent policies is their distribution of play, which is subject to change.

Before we can start addressing this question, we must first clarify its exact nature[1]. Assume we are trying to solve a game Γ using a learning-based game-solving algorithm. The empirical game evolves iteratively, as policies are calculated and added. New policies are determined by computing an ABR in relation to a strategic context, which is defined by the current state of the empirical game. This context is dictated by the strategies σ of all coplayers $-i$, which in turn are distributions over their respective policies ($\pi \in \Pi$).

Returning to our transfer question, it asks us to compute an ABR using minimal samples of experiences from the game. Any artifacts generated from previous ABR computations can be used freely in the current ABR computation. This establishes our Sequential-Response Transfer problem stated fully in Problem 1.

Problem 1 (Sequential-Response Transfer)**.** *Consider a game* Γ *and a series of* c *strategic contexts defined by the coplayers' strategies* $\sigma^0_{-i}, \dots, \sigma^c_{-i} \in \Delta(\Pi_{-i})$*. Given:*

- *the current strategic context* σ^t_{-i}*, and*
- *any artifacts produced by computing best-responses to the previous* $[0, t-1]$ *contexts,*

[1]Chapter 2 provides full definitions of the concepts briefly introduced here.

compute a best-response to σ^t_{-i} using the fewest samples of experience from the game.

The many contexts that can be transferred from are a boon and a bane. On one hand, they provide the opportunity to maximize knowledge reuse. On the other, they introduce significant complexity into the transfer operation. To facilitate our analysis of the solution, we simplify the problem through logical induction.

Consider that, across ABR calculations, the size of the players' strategy sets only increases. Consequently, elements to learn are exclusively added, never subtracted. This simplification enables us to reduce the problem to the evaluation of what can be transferred across a single step of response calculations. This simplified scenario is referred to as the one-step response transfer problem (Problem 2).

Problem 2 (One-Step Response Transfer)**.** *Consider a game Γ and the two strategic contexts $\sigma^0_{-i} \in \Delta(\hat{\Pi}^0_{-i})$ and $\sigma^0_{-i} \in \Delta(\hat{\Pi}^0_{-i})$, where $\hat{\Pi}^0_{-i} \subseteq \hat{\Pi}^1_{-i} \subseteq \Pi_{-i}$. Given:*

- *the current strategic context σ^1_{-i}, and*
- *any artifacts produced by computing a best-response to σ^0_{-i},*

compute a best-response to σ^1_{-i} using the fewest samples of experience from the game.

The one-step problem is able to fully substitute for the sequential problem only if all coplayers play in full support in every context. This is not typically the case, and instead, it is common for policies to come in and out of support through game solving. The one-step problem instead allows us to focus directly on the problem of what is transferable and how to transfer it. Moreover, as discussed below, intermediate solutions can be deployed to extend one-step solutions to sequential solutions.

In this dissertation, I utilize these problem statements and their instances to pinpoint and structure opportunities for transfer learning. From these opportunities, I will categorize transferable knowledge and formulate algorithms to disseminate it. Solutions to these problems can lessen the expense of each ABR calculation, consequently reducing the overall cost incurred by learning-based game-solving algorithms. In total, showing the benefit of incorporating transfer learning in game solving.

1.3 Knowledge Taxonomy

As elucidated above, the sequence of ABR computations performed in game solving share common elements in the problems they pose. These elements and their structure mirror steps in the process of constructing an empirical game. It is this structure that I leverage to define the basic units of knowledge that can be acquired and subsequently transferred. I employ Ring [2013]'s definition of knowledge as predictions. This section introduces a taxonomy of such knowledge, as illustrated in Figure 1.5. Throughout the discussion, I periodically revisit our market example to provide context for the different types of knowledge.

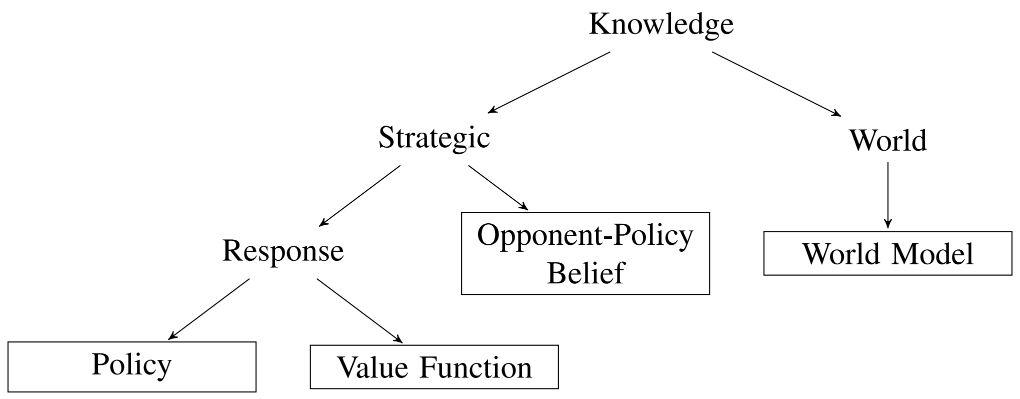


Figure 1.5: **Taxonomy of transferable knowledge.** Boxed items denote types of knowledge representations that can be instantiated.

1.3.1 Strategic Knowledge

The initial type of knowledge is termed *strategic knowledge*, encompassing the potential strategic interactions between players. This knowledge represents predictions that are dependent on the strategies chosen by all players.

In the simplest case, an agent can assume that its coplayers will each play a fixed strategy. An agent can leverage this assumption to inform predictions, which are then utilized to devise an appropriate response. I term this *response knowledge* and it encodes predictions that are only valid in the context of its assumed coplayers' strategies. In the market, each vendor entered the market with a fixed policy offering select goods and willingness to sell them at predefined prices[2]. From this assumption of fixed coplayers, we generated response knowledge that allowed us to predict where to find the best bargain for each good. Enabling us to plan an efficient shopping trip before arriving at the market.

Response knowledge can be represented by various methods. The focus in this dissertation, however, is on RL-specific representations, specifically the *policy* and *value function*. A policy is a mapping from an agent's observations to a distribution of their actions. A value function estimates the expected cumulative discounted reward, or return, anticipated from being in a particular state.

[2]Vendors may actually have significantly more complicated policies that are stochastic or change preferences throughout the day. As the analysis stays consistent, I focus here on a simplified example.

Action-value functions offer a more specific estimate, by additionally assuming a specific action was selected. Value functions can be used to induce a policy, for example, by selecting actions with the highest expected return.

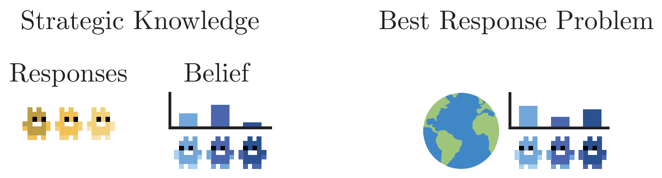


Figure 1.6: **Outline of strategic knowledge iconography.** Coplayer policies and their respective response are paired by equivalent saturation in colors. Beliefs are bar graphs portraying the likelihood of each coplayer policy.

Every policy that a coplayer may use is an opportunity to construct corresponding response knowledge. What happens, though, if the coplayer randomizes their play by sampling a policy from a distribution? There is now uncertainty in the coplayer's policy, and therefore, uncertainty in the appropriate response. To manage this, an agent might either explicitly or implicitly predict the coplayer's policy based on their interactions, and utilize this prediction to guide their response behavior. I refer to the agent's prediction of the coplayer's current policy as its *opponent-policy belief knowledge*. Note here that while the term opponent is used for historical purposes, belief knowledge can be maintained over coplayers generally. The agent interprets this belief as a distribution over the opponent's policies, using in-game observations as evidence to shape this belief. With a precise belief, the agent can then select the appropriate response.

To illustrate belief knowledge, let's revisit the marketplace scenario. Imagine one of the vendors occasionally has a sale on their goods. Your typical response to their regular prices is to bypass their stall; however, during a sale, you would actually prefer to buy their goods. However, you cannot determine if they are having a sale unless you visit the stall. Consequently, you can form a belief about the probable occurrence of a sale and use that to decide whether it's worth checking.

In Figure 1.6, I provide an overview of the iconography that will be used for strategic knowledge as it relates to ABR learning problems. In Part II, I investigate applications of strategic knowledge in both transfer learning and game solving.

1.3.2 World Knowledge

The second type of knowledge is *world knowledge*[3]. This knowledge is composed of predictions that are independent of the players' strategies. As the term suggests, world knowledge mainly involves the agent's capability to predict how the world will react to specific actions. These predictions materialize as models of transition dynamics or the operational mechanisms of the system. Figure 1.7 depicts the iconography I will use for world models, note that the predicted successor state of the world is courser and erroneous, mimicking errors in prediction. In our market example, this is exactly the market's mechanisms: the vendor selection mechanism, the shared currency and its exchange methods, the layout of the market, etc..

A significant aspect of these dynamics is the reward signal for all players. The reward signal is a unique component of an agent's observation that measures the impact of their actions. However, it's critical to understand that rewards may not be immediate; they can instead reflect the effectiveness of a previous action or a sequence of actions. The discrepancy between actions and their corresponding rewards is known as the credit assignment problem.

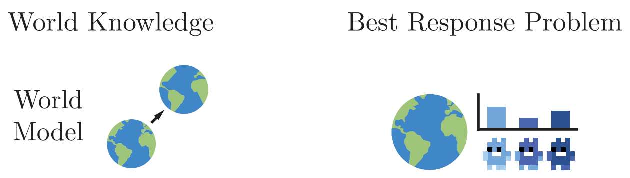


Figure 1.7: **Outline of world knowledge iconography.** The world model makes predictions of successor states that may introduce modeling errors. World models typically condition their prediction on action(s) from the agent(s), which is suppressed in this icon.

Learning world dynamics is no easy task, because players often only have a limited view of the world. Outside of their view, other players may be taking actions, or unseen consequence of a player's own action may be realized. This leads to a key problem of multiagent learning that is disentangling each player's responsibility for changes in the world. To side-step this problem, a key assumption I will make throughout this dissertation is access to the coplayer's actions and observations throughout training. Importantly, however, all policies will not depend on on this extra information during evaluation.

In summary, we have now separated the components of knowledge that an agent must learn to solve a multiagent system. Now we can explore methods that exploit this known structure in order to more efficiently solve multiagent systems. Throughout this dissertation, I will do just that by proposing new algorithms for transferring each type of knowledge.

[3]Hereafter, I use the term *world* to refer to the system in which the agents interact. This terminology is deliberately selected to differentiate from alternative terms such as *environment*, which suggests a single-agent system, and *game*, which I will attempt to exclusively use for empirical games given its already overloaded usage.

1.4 Thesis Statement

After laying down both the game-solving paradigm we intend to utilize and our categorization of transferable knowledge, we can formulate the principal question this dissertation aims to answer:

How can we lessen the experiential cost of learning-based game-solving algorithms?

As an answer to this question, this dissertation puts forward the following thesis:

THESIS

In learning-based game-solving algorithms, the response learning problems exhibit a common structure that reflects the empirical-game building process, thus facilitating the transfer of knowledge from previous responses and consequently reducing the experiential cost of game solving.

To elaborate, this dissertation develops algorithms designed to minimize the experiential cost associated with learning-based game-solving algorithms by employing methods of transfer learning. These algorithms exploit the fact that the response learning problems share common elements that reflect the steps of building an empirical game. Inspired by this structural similarity, I devise a corresponding taxonomy of transferable knowledge. Subsequently, I create a suite of algorithms designed to facilitate the transfer of this knowledge within the context of learning-based game-solving algorithms.

1.5 Outline & Contributions

Part I: Preliminaries The first part of the dissertation provides the necessary background required for the remaining parts. It covers fundamental concepts from both the fields of machine learning and game theory.

Part II: Strategic Knowledge Transfer In Part II, I address the learning of strategic knowledge. I begin by introducing the opponent-mixture transfer problem (Chapter 3). This problem seeks to generalize an agent's knowledge across varying distributions of the same set of coplayers. Chapter 4 reveals that response policies to each coplayer cannot be directly transferred across coplayer distributions. Consequently, additional structure is necessary for generalizing response policies. In Chapter 5, I demonstrate that value-based response policies provide this generalizing capability.

After establishing a method for transferring response knowledge, I explore refinements to this approach. Chapter 6 considers a suite of opponent-policy likelihood models that form a belief in the coplayer's current policy. This belief further informs selecting the appropriate response. Finally, I propose two algorithms that apply strategic knowledge transfer to game solving. The Mixed-Oracles algorithm illustrates how transfer can focus learning solely on new coplayer policies. The Mixed-Opponents algorithm shows how transfer can serve as a heuristic to guide the discovery of new policies within the game. Compared to methods without transfer, both strategies demonstrate lower experiential costs and potentially yield stronger solutions. The contributions of this part of the dissertation appear previously within:

Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Learning to play against any mixture of opponents. *Frontiers in Artificial Intelligence*, page to appear, 2023a

Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Iterative empirical game solving via single policy best response. In *9th International Conference on Learning Representations*, 2021

Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Strategic knowledge transfer. *Journal of Machine Learning Research*, 24:to appear, 2023b

Part III: World Knowledge Transfer In Part III, I delve into the study of the learning of world knowledge. I first demonstrate that the co-learning of an empirical game and a world model offer reciprocal advantages. Chapter 9 reveals that empirical games provide a diverse perspective on potential strategies, thereby informing a more general world model. Chapter 10 shows that through planning world models can reduce the experiential cost of learning new policies to expand the empirical game. These benefits merge in the creation of a new game-solving algorithm, Dyna-PSRO, that lowers experiential cost through the transfer of world knowledge (Chapter 11). The contributions of this part of the dissertation appear previously within:

Max Olan Smith and Michael P. Wellman. Co-learning empirical games and world models. *CoRR*, 2023. URL https://arxiv.org/abs/2305.14223

Part IV: Conclusion I conclude the dissertation by summarizing how the previous sections uphold the thesis of this work. Moreover, I discuss future avenues of research inspired by this dissertation.

Figure 1.8 provides a graphical representation of this dissertation, emphasizing the primary algorithmic contributions and novelties.

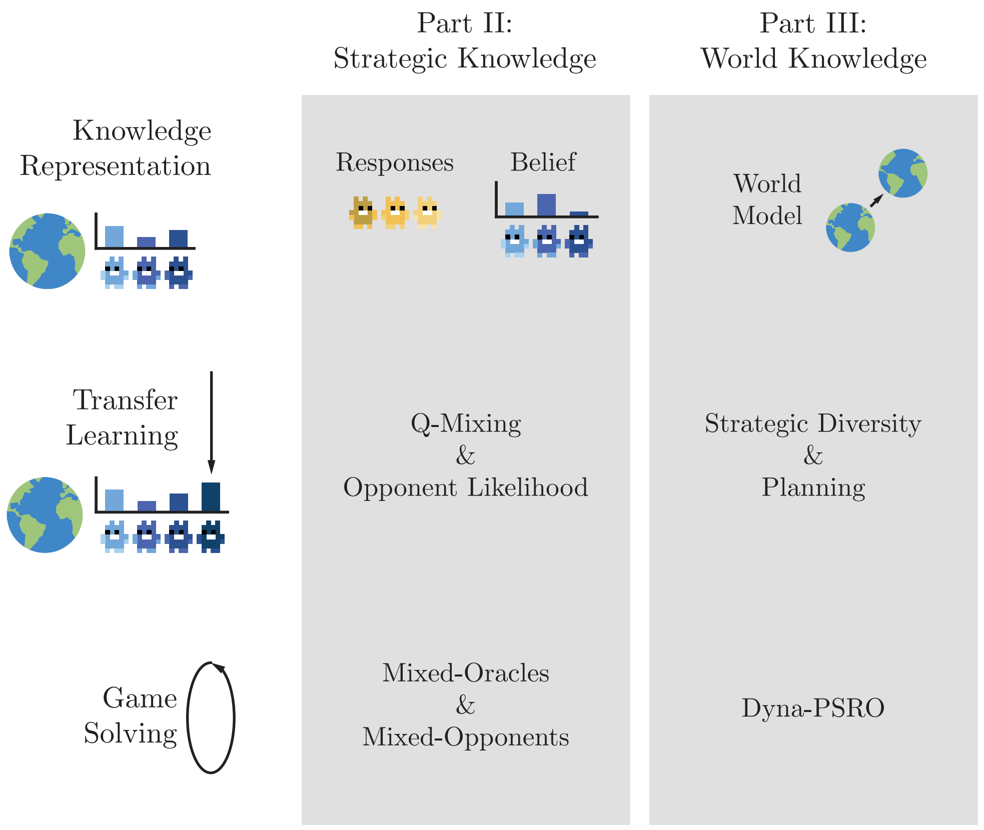


Figure 1.8: **Outline of this dissertation.** The contributions of the dissertation are split into two primary parts: strategic and world. Each part begins with the development of its respective representation of knowledge. Subsequently, transfer learning algorithms are devised for the one-step transfer case. Finally, game solving algorithms are introduced that continually transfer knowledge for more efficient game solving.

CHAPTER 2

Background

In this chapter, I provide an overview of the relevant techniques for resolving multiagent systems. Since this dissertation straddles the fields of RL and GT, I start by defining the pertinent concepts from both areas. Section 2.1 kicks off by introducing the RL problem. In the traditional RL approach, agents are trained without explicit regard for the other agents in the system. Section 2.2 reviews adaptations to the RL problem that take coplayers into account. These distinct approaches constitute the field of Multiagent Reinforcement Learning (MARL). MARL methods typically aim to train efficient agents to play in a specified set of strategic contexts (often just one). Alternatively, Game Theory, introduced in Section 2.3, is concerned with play not constrained to any specific context. In Section 2.4, I introduce the field of EGTA to demonstrate how GT can be applied to complex games. In the following section, Section 2.5, I take a detour and discuss the related work in the field of transfer learning. Finally, in Section 2.6 I provide extensive details of the more complex games that are used to evaluate our game solving algorithms.

2.1 Reinforcement Learning

RL is a computational method for learning through interaction [Sutton and Barto, 2018]. Learning is achieved by an agent situated within a world, aiming to accomplish a goal. Progress towards this goal is marked by the reception of reward. Unique to RL is that the agent learns from *interaction* data, and the agent's decisions impact the generation of this data in the future. This demands that the agent *explore* diverse behaviors to achieve their goal. This distinction sets RL apart from supervised learning, where the learner has analogous access to the optimal behaviors (i.e., "labels").

2.1.1 Interaction Loop

To begin defining methods for learning through interaction, we must begin by defining what we are interacting with. This quantity has been referred to primarily as the *environment* or *world*. Within this dissertation, I reserve the word *environment* to refer to single-agent systems (i.e., worlds/games with one player). I correspondingly use world to refer to the non-player part of the game, the main purpose of this is to reduce overloading the term game.

The primary class of environments studied within RL are those that can be represented by a Markov decision process (MDP) [Puterman, 1994]. MDPs are mathematical models of decision processes with discrete-time and stochastic-control. MDPs also feature the Markov property, meaning that they are memoryless so a decision-maker need only base their decision on the information currently available [Markov, 1954].

Interaction with an MDP begins by the sampling of the environment's initial state $s^0 \sim d^0$, where $d^0 \in \Delta(\mathcal{S})$ is a distribution over the environment's initial state. Generally, the environment at discrete time $t \in \mathcal{T}$ exists in *state* $s^t \in \mathcal{S}$. The agent perceives the environment through its sensors and then takes an *action* $a^t \in \mathcal{A}(s^t)$ using its actuators. In other environment definitions, the sensors and actuators may respectively limit the information received about the environment or the agent's ability to influence the current state. However, we ignore these complications for now and assume that agent perceives the environment's true state s^t and has access to all available actions $\mathcal{A}(s^t)$. $\mathcal{A}(s^t)$ may be denoted $\mathcal{A}$ when the referred state is unambiguous or the available actions are the same across all states.

An agents rule for selecting an action for a state is a *decision*, and the collection of decisions for all states constitute the agent's *behavior*. There are many possible ways to characterize behavior. A *policy* generally prescribes an agent's behavior in all states:

$$\pi : \mathcal{S} \to \Delta(\mathcal{A}). \tag{2.1}$$

A policy may be stochastic mapping to $\Delta(\mathcal{A})$, or deterministically select an action.

The agent's action alter the state of the environment driving it into a potentially new state. This process is referred to as the *transition dynamics* or *dynamics* of the environment $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. $p(x)$ describes a general probability mass function $\mathbb{P}(X = x)$ for the random variable taking on values $x \in \mathcal{X}$. Similarly, capital non-stylized characters are left to refer to random variables (e.g., X). This particular choice of notation is motivated by the variety of forms used to describe the environment's dynamics.

In addition to the agent now perceiving the new state they also receive a *reward* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that is a signal for the goodness of choosing the previous action. The space of rewards often doesn't cover $\mathbb{R}$, and is typically discrete, so we use $\mathcal{R}$ to represent the subspace of rewards defined by the environment. Notation is also abused to have r refer to both the reward-function and a single reward, because we will not focus on reward-functions directly through this dissertation. For this same reason, we will often combine the dynamics and reward function as

$$p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S}) \times \Delta(\mathcal{R}), \tag{2.2}$$

for ease of notation when no distinction between the two components is required. A discount factor $\gamma \in [0, 1]$ also indicates an agent's preference for short- or long-term rewards.

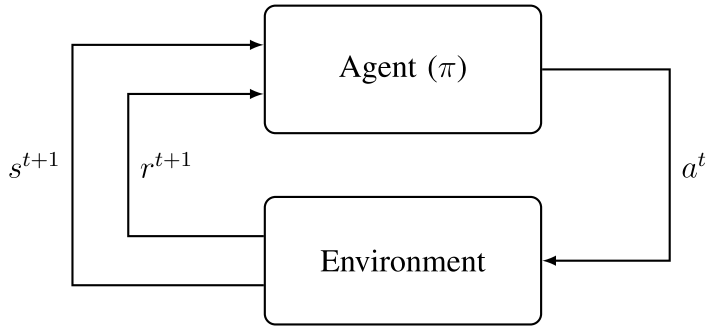


Figure 2.1: **Agent-environment interaction loop.**

That completes the description of the primitive elements of an MDP. Additional concepts are built upon these elements for composite reasoning.

The *interaction protocol* governs the order and frequency of which the agent and environment respectively make decisions or update. The assumed protocol is simplest one where the agent and environment alternate acting and transitioning respectively. The interaction begins in the environment's *initial state* and continues for a number of interactions referred as the *horizon*. The horizon is typically assumed to be finite, meaning that the interaction ends at a *terminal state*, instead of continuing in perpetuity. A round of interaction $(s^t, a^t, r^{t+1}, s^{t+1})$ is referred to an agent's experience or as a transition. Sequences of transitions are called a *trajectory* τ. Trajectories that begin in an initial state and end in a terminal state are called an *episode*. Experiences and trajectories are the primary units of data used to train reinforcement learners. As collecting an individual experience is often costly, counts of experiences serve as a method for evaluating the cumulative cost of a reinforcement learner's learning algorithm.

Often an agent's sensors may not be able to fully capture the state of the environment. Partial representation of the state of the environment are referred to as an *observation* $o^t \in \mathcal{O}$ of the underlying state. Analogous definitions of the composite terms we have introduced previously can be made with observation histories substituting as states. Transitioning to observations often forfeits or relaxes any theoretical guarantees about a learning algorithm.

2.1.2 Value Functions

The agent's goal is to maximize its received cumulative discounted reward. This quantity, termed *return*, is defined as:

$$G^t \equiv \mathbb{E}_\pi \left[\sum_{k=t}^{\infty} \gamma^{k-t} r(s^k, a^k) \;\middle|\; \pi \right]. \tag{2.3}$$

Note here that the return depends on the policy. This is because future rewards are dependent on the actions selected by the policy.

A *value function* is an estimate of the agent's return for a specific state. There are two main value functions: the value function $V_\pi : \mathcal{S} \to \mathbb{R}$, defined formally as:

$$V_\pi(s) \equiv \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma V_\pi(s') \right]. \tag{2.4}$$

It is also common to measure the return expected conditional on an action. This is characterized as the *action-value function* $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for a fixed policy π. As shorthand, this function can be referred to as the Q-value function or Q function. The action-value function is defined using the value function (Equation 2.4) as follows:

$$Q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma V_\pi(s') \right]. \tag{2.5}$$

A policy can be induced from a value or action-value function by defining a method for action selection from values. The polic(ies) that exhibit the highest possible return are said to be optimal, and their corresponding value functions are marked with $*$, as in Q^*, V^*. For action-values, the simplest induced policy is the *greedy* policy that selects the action achieving the highest return in every state:

$$\pi_Q(a \mid s) = \mathbb{1} \left\{ a = \arg\max_{a'} Q(s, a') \right\} \tag{2.6}$$

As, during learning, there is impetus to explore behaviors, it is common to use an ϵ-*greedy* policy. This policy behaves greedily $1 - \epsilon$ proportion of the time, and randomly the other ϵ proportion of the time:

$$\pi_{Q,\epsilon}(a \mid s) = \begin{cases} 1 - \epsilon & a = \arg\max_{a'} Q(s, a'), \\ \frac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise.} \end{cases} \tag{2.7}$$

2.1.3 Q-Learning

Value-based RL algorithms are the subclass of algorithms that learn a value or action-value function in the process of learning a policy. The value function can be learned in isolation and a policy induced from it post-learning (e.g., greedy). Or, the value function can be learned in tandem with a direct representation of the policy.

Q-learning [Watkins, 1989, Watkins and Dayan, 1992] is one class of value-based RL algorithms for learning optimal Q-value functions from trajectories. They work by maintaining an current estimate of the Q-values and iteratively making refinements to the estimates from experiences. Notably, this class of algorithms is *off-policy*, meaning that the experiences may be generated from any policy and used to learn the optimal Q-value function. The policy used to generate data for learning is called the *behavioral* policy. In Q-learning, a standard choice of behavioral policy is the ϵ-greedy policy induced by the current Q-values.

Tabular Q-Learning. A simple version of Q-learning is called *tabular* Q-learning. In tabular Q-learning, the Q-values are stored in a table indexed by state-action pairs. Due to the nature of this indexing, tabular Q-learning is limited worlds with small and discrete state and action spaces. The entries of the table are updated as a linear combination of the current estimate and the greedy value estimate:

$$\begin{aligned} Q(s,a) &\leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a')) && (2.8) \\ &= Q(s,a) + \alpha \cdot \left[\underbrace{r + \gamma \cdot \max_{a'} Q(s',a')}_{\text{target}} - Q(s,a) \right], && (2.9) \end{aligned}$$

where α is the algorithm's learning rate. The behavior that an RL algorithm is updating towards is referred to as the *target* behavior. In the case of tabular Q-learning it can be shown that a greedy target converges to the optimal Q-values.

Deep Q-Learning. Due to the combinatorial nature of the joint state-action space, Q-value functions typically cannot be implemented with a table. Deep Q-learning [Mnih et al., 2015, Tsitsiklis and Van Roy, 1997] addresses this problem by using deep neural network (DNN) [Goodfellow et al., 2016] as its Q-value function. DNNs are nonlinear function approximators that learn hierarchies of features across the layers of the neural network. The resulting DNN-based Q-value function learns jointly to project the joint state-action space into a compact representation. Q-values are jointly learned to be associated with these representations. The Deep Q-Network (DQN) [Mnih et al., 2015] was the first agent implementation to feature large wins for this method on the Atari game suite [Bellemare et al., 2013]. Paramount to the success of DQN are following methodological details:

- *Replay buffer*: Transitions are stored in a replay buffer [Lin, 1992]. Batches of transitions are sampled from the replay buffer and used to update the network. By varying the settings of the replay buffer, such as its size and sampling method, a practitioner can adjust the correlation between transitions in a batch.

- *Target network*: The Q-values used to compute the target are produced by a DNN with older parameters. Keeping these parameters fixed for a short window creates a more stable training objective. It also mitigates issues where Q-values would rapidly increase across updates.

Let θ be the current parameters, and θ^- be the target's parameters. Parameterized functions have their associated parameters marked as a subscript f_θ or as a conditional input $f(\cdot \mid \theta)$. Then we can rewrite the loss function from tabular Q-learning as:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{s,a,r,s'}\left[(r + \gamma \cdot \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a))^2\right] & (2.10)\\ &= \mathbb{E}_{s,a,r,s'}\left[(y - Q_\theta(s, a))^2\right], & (2.11)\end{aligned}$$

where $y = r + \gamma \cdot \max_{a'} Q_{\theta^-}(s', a')$ is the target. From this loss function we can derive an update rule by gradient descent as:

$$\begin{aligned}\nabla_\theta \mathcal{L} &= \mathbb{E}_{s,a,r,s'}\left[(y - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)\right] & (2.12)\\ \theta &\leftarrow \theta - \eta \nabla_\theta \mathcal{L}. & (2.13)\end{aligned}$$

This update is applied periodically from batches sampled from a replay buffer. Periodically θ^- is updated to be closer to θ. This is commonly done by simply copying the parameters, or computing an average between the parameters between the current and target parameters.

Double DQN. While the incorporation of a target network in DQN mitigates escalating Q-values, this issue can still persist due to the algorithm's sensitivity to hyperparameter selection. van Hasselt et al. [2016] proposed decoupling target estimation and action selection during learning:

$$y = r + \gamma \cdot Q_{\theta^-}(s', \max_{a'} Q_\theta(s', a')), \tag{2.14}$$

where y is again the target and is substituted into Equation 2.12. This implementation trick has empirically been shown to reduce value overestimation.

2.1.4 Policy Gradient

Alternative to learning a value function and using it to construct a policy, a policy can be directly learned. These *policy gradient* algorithms directly estimate a return gradient with respect to the policy [Sutton et al., 1999a]. Updating the policy's parameters following the policy gradient increases the probability of an action proportional to the return it received. Let $\pi_\theta(a \mid s)$ be the policy π parameterized with θ, we write the policy gradient as:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t \cdot r_t\right] = \mathbb{E}_{\pi_\theta}\left[\sum_t \sum_a \nabla_\theta \pi_\theta(a \mid s_t) \cdot Q^{\pi_\theta}(s_t, a)\right] \tag{2.15}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_t \sum_a \frac{\nabla_\theta \pi_\theta(a \mid s_t)}{\pi_\theta(a \mid s_t)} \cdot Q^{\pi_\theta}(s_t, a) \cdot \pi_\theta(a \mid s_t)\right] \tag{2.16}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_t \sum_a \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot Q^{\pi_\theta}(s_t, a) \cdot \pi_\theta(a \mid s_t)\right] \tag{2.17}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_t \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot Q^{\pi_\theta}(s_t, a_t) \mid s_t\right]\right] \tag{2.18}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot Q^{\pi_\theta}(s_t, a_t)\right]. \tag{2.19}$$

The first two steps involve multiplying in the identity $\pi_\theta(a \mid s_t)/\pi_\theta(a \mid s_t)$ and performing the likelihood ratio trick [Glynn, 1990]. The next step comes from observing that the inner summand $\sum_a \ldots \cdot \pi_\theta(a \mid s_t)$ is the definition of the expectation over the action space. Finally, the inner expectation can be lifted into the outer expectation by the law of iterated expectations $\mathbb{E}[\mathbb{E}[Y \mid X]] = \mathbb{E}[Y]$. Applied to our derivation, this states that the outer expectation that averages over all states and timesteps is equivalent to first averaging over both states and actions (and all timesteps), without conditioning on the current state[1].

Williams [1992] first introduced policy gradients with the REINFORCE algorithm that uses return samples $R_t = \sum_{k=t}^{\infty} \gamma^{k-t} \cdot r_k$ as a value estimate:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t \cdot r_t\right] = \mathbb{E}_{\pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot R_t\right]. \tag{2.20}$$

Baseline. Directly estimating the return R_t can have high variance depending on the stochasticity of the policy and environment. It is possible to reduce the variance of this estimate using a state-dependent *baseline* $b(s_t)$. Crucially, the baseline must only be state-dependent, because the lack of action dependence allows it to not modify the policy gradient:

$$\sum_a \nabla_\theta \pi_\theta(a \mid s_t) \cdot b(s_t) = b(s_t) \nabla_\theta \sum_a \pi_\theta(a \mid s_t) = b(s_t) \nabla_\theta 1 = 0. \tag{2.21}$$

[1]The law of iterated expectations requires that the outer variable s_t is independent of the inner expectation a, which we get through assuming the policy is Markovian.

A reasonable choice for the baseline is the state-value function V^{π_θ}. This choice modifies our return estimate with an estiamte of how much *advantage* an action has over the expected action:

$$\begin{aligned} A(s_t, a_t) &= Q(s_t, a_t) - V(s_t) && (2.22) \\ &\approx R_t - V(s_t). && (2.23) \end{aligned}$$

The value function can be learned alongside the policy in a joint optimization:

$$\begin{aligned} \mathcal{L}^{pg} &= \mathbb{E}_{\pi_\theta}\left[\mathcal{L}^{pg}_{policy} + \lambda_{value} \cdot \mathcal{L}^{pg}_{value}\right] && (2.24) \\ \mathcal{L}^{pg}_{policy} &= -\log \pi(a_t \mid s_t) \cdot (R_t - V_\theta(s_t)) && (2.25) \\ \mathcal{L}^{pg}_{value} &= \frac{1}{2}\|V_\theta(s_t) - R_t\|^2, && (2.26) \end{aligned}$$

where $V_\theta(s_t) \approx V^{\pi_\theta}(s_t)$, and λ_{value} weights the relative importance of the value objective compared to the policy objective. Therefore, if an action a_t offers an advantage the probability will correspondingly increase, and decrease (or remain the same) otherwise.

Actor-Critic. The combination of learning a policy and a value function is referred to as an *actor-critic* algorithm. Here, the policy is the actor generating experiences, and the value function is the critic offering evaluations of the actor's decisions. Actor-critic methods afford an additional means to reducing the variance of the policy gradient estimate through bootstrapping. To apply bootstrapping, consider splitting the full return estimate R_t into two partitions. The first term, $R_{t:t+n}$, continues to be a sampled return, but is truncated after n timesteps. The remainder of the full return estimate comes from bootstrapping off the value function $V_\theta(s_{t+n})$. Giving us the n-step actor-critic objective:

$$\begin{aligned} \mathcal{L}^{ac} &= \mathbb{E}_{\pi_\theta}\left[\mathcal{L}^{ac}_{policy} + \lambda_{value} \cdot \mathcal{L}^{ac}_{value}\right] && (2.27) \\ \mathcal{L}^{ac}_{policy} &= -\log \pi(a_t \mid s_t) \cdot (R^n_t - V_\theta(s_t)) && (2.28) \\ \mathcal{L}^{ac}_{value} &= \frac{1}{2}\|V_\theta(s_t) - R^n_t\|^2 && (2.29) \\ R^n_t &= \sum_{k=0}^{n=1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\theta(s_{t+n}). && (2.30) \end{aligned}$$

Bootstrapping off of V_θ introduces biases proportional to its discrepancy with the true value function. However, empirically n-step actor-critic has demonstrated better sample-efficiency than 1-step actor-critic.

Entropy. As we saw in value-based RL, encouraging exploration is crucial to prevent early convergence to a suboptimal policy. In policy gradient methods, a common exploration strategy is to include a policy-based entropy term:

$$\mathcal{H}(\pi(s_t)) = -\sum_a \pi(a \mid s_t) \log \pi(a \mid s_t). \tag{2.31}$$

This term can be used to encourage the policy to uniformly sample actions. Applied to the actor critic objective we previously defined leaves us with:

$$\mathcal{L}^{ac} = \mathbb{E}_{\pi_\theta}\left[\mathcal{L}^{ac}_{policy} + \lambda_{value} \cdot \mathcal{L}^{ac}_{value} + \lambda_{entropy} \cdot \mathcal{L}^{ac}_{entropy}\right] \tag{2.32}$$

$$\mathcal{L}^{ac}_{policy} = -\log \pi_\theta(a_t \mid s_t) \cdot (R^n_t - V_\theta(s_t)) \tag{2.33}$$

$$\mathcal{L}^{ac}_{value} = \frac{1}{2}\|V_\theta(s_t) - R^n_t\|^2 \tag{2.34}$$

$$\mathcal{L}^{ac}_{entropy} = \mathcal{H}(\pi(s_t)) \tag{2.35}$$

$$R^n_t = \sum_{k=0}^{n=1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\theta(s_{t+n}). \tag{2.36}$$

Importance Weighted Actor-Learner Architecture (IMPALA). The final detail of policy gradient methods that is concerned within this dissertation concerns scaling learning the algorithm onto many devices. RL algorithms are often bottlenecked on their experience generation, and not on the throughput of data through the learner (the process performing the gradient updates). Therefore, a natural solution to this problem is to increase the number of processes generating data for the learner. A challenge with this solution is that the behaviors generating data may follow a different policy than the learner, making the learning problem *off-policy*. We concern ourselves with the class of solutions integrating this notion with the actor-critic algorithm, the so-called Asynchronous Advantage Actor Critic (A3C) algorithm [Mnih et al., 2016]. In it, many asynchronous *behavioral* policies μ generate data used to learn on a *target* policy π. Differences between the behavioral and target policies can be corrected through importance sampling (IS). One such effective method for this correcting is the V-trace operator first introduced in the Importance Weighted Actor-Learner Architecture (IMPALA) [Espeholt et al., 2018]. The n-step V-trace target for $V(s_t)$ is as follows:

$$v_s \equiv V(s_t) + \sum_{t=s}^{s+n-1} \gamma^{t-1} \cdot \left(\prod_{i=s}^{t-1} c_i\right) \cdot \delta_t V, \tag{2.37}$$

where,

$$\delta_t V \equiv \rho_t \cdot (r_t + \gamma \cdot V(s_{t+1}) - V(s_t)), \tag{2.38}$$

is a temporal difference for V, and

$$\rho_t \equiv \min(\bar{\rho}, \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)}) \qquad c_i \equiv \min(\bar{c}, \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)}), \tag{2.39}$$

are truncated importance sampling weights (assuming $\bar{\rho} \geq \bar{c}$). Applying V-trace to A3C gives us the IMPALA objective:

$$\mathcal{L}^{impala}_{policy} = -\log \pi_\theta(a_t \mid s_t) \cdot (r_t + \gamma \cdot v_{s+1} - V_\theta(s_t)), \tag{2.40}$$

$$\mathcal{L}^{impala}_{value} = \frac{1}{2} \| V_\theta(s_t) - v_s \|^2, \tag{2.41}$$

$$\mathcal{L}^{impala}_{entropy} = \mathcal{H}(\pi(s_t)), \tag{2.42}$$

$$\mathcal{L}^{impala} = \mathbb{E}_{\pi_\theta} \left[\mathcal{L}^{impala}_{policy} + \lambda_{value} \cdot \mathcal{L}^{impala}_{value} + \lambda_{entropy} \cdot \mathcal{L}^{impala}_{entropy} \right]. \tag{2.43}$$

(2.44)

2.1.5 Model-Based RL

Model-Based RL (MBRL) algorithms construct or use a model of the environment (henceforth, *world model*) in the process of learning a policy or value function [Sutton and Barto, 2018]. World models refer to anything that can be used to predict how an environment will respond to a hypothetical action. For example, world models may either predict successor observations directly (e.g., at pixel level [Wahlström et al., 2015, Watter et al., 2015]), or in a learned latent space [Ha and Schmidhuber, 2018a, Gelada et al., 2019, Ha and Schmidhuber, 2018b]. World models may also learn the system's reward function independently or alongside a predictor of the successor observations.

The process of utilizing a world model to generate or enhance a policy is termed as *planning*. There are two subtly different but significant types of planning for our discussion in Part III.

Decision-time planning is a process that uses a world model to develop or refine a policy for the agent's current state, informing its next actionable step. This kind of planning is traditionally associated with the term planning. It also bears similarity to search and, hence, is sometimes referred to as lookahead search. For instance, consider an agent π in state s^t with a world model w. Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006] is one instance of decision-time planning. In MCTS, an agent samples numerous hypothetical rollouts of the future utilizing its policy and the world model,

$$A^t \sim \pi, S^{t+1} \sim w, R^{t+1} \sim w, A^{t+1} \sim \pi, \ldots, S^T \sim w, R^T \sim w,$$

with T representing the terminal state (therefore, the environment is episodic). The sampled returns for all actions noted as A^t are averaged per action, and the action with the highest return is selected. Decision-time planning can also be integrated with learning, by using planned values directly or indirectly in learning [Silver et al., 2017b, Oh et al., 2017].

Background planning, conversely, is planning that can transpire in any state and does not directly influence action selection in the real world. This introduces the question of which states should be prioritized by the agent for planning. The response to this question is known as the agent's method of search control. Search control establishes the initial state distribution for background planning, and rollouts can be accomplished by iteratively sampling between a policy and world model. However, unlike in decision-time planning, we are not directly using planning to determine a real-world action in background planning. So, how does background planning improve the policy? It does so by storing *planned experiences* and learning from them as if they were real-world experiences. This process should indirectly enhance the policy for any states sampled by the search control procedure.

Model-based methods are not without fault. Talvitie [Talvitie, 2014] demonstrated that even in small Markov decision processes (MDP) [Puterman, 1994], model-prediction errors tend to compound—rendering long-term planning at the abstraction of observations ineffective. A follow-up study demonstrated that for imperfect models, short-term planning was no better than repeatedly training on previously collected real experiences; however, medium-term planning offered advantages even with an imperfect model [Holland et al., 2018]. Parallel studies hypothesized that these errors are a result of insufficient data for that transition to be learned [Kurutach et al., 2018, Buckman et al., 2018]. To remedy the data insufficiency, ensembles of world models were proposed to account for world model uncertainty [Buckman et al., 2018, Kurutach et al., 2018, Yu et al., 2020], and another line of inquiry used world model uncertainty to guide exploration in state-action space [Ball et al., 2020, Sekar et al., 2020].

2.2 Multiagent Reinforcement Learning

RL algorithms generally take no explicit consideration of the presence of coplayers. Nevertheless, it is possible to directly apply them to games. This is by done by viewing the game as a black-box with no distinction between coplayers and the world. I review now relevant MARL algorithms, which are RL algorithms that explicitly consider coplayers in either their design or motivation.

The notation that we previous applied to single-agent RL is similarly defined in MARL settings. I will now reserve subscripts for denoting player identities such as player 1: π_1, and therefore, will be moving the notation of time to superscripts (e.g., player 2's state at timestep 42 is s_2^{42}). The joint of an element across all agents is represented with a boldface character $\mathbf{a} = (a_i^t, \ldots, a_{n-1}^t)$ (for n players). Negated subscripts represent the joint across all other agents. For example, all but player i's action is denoted a_{-i}.

2.2.1 Joint-Action Learners

Claus and Boutilier [1998] demonstrated that learning the values of joint-actions as opposed to only the ego-agent's actions reduced the variance in value estimates by controlling for these unobserved confounders. They fittingly called their approach Joint-Action Learner (JAL), and it learns the following joint-action value function:

$$Q(s_i, \boldsymbol{a}) = Q(s_i, a_i, a_{-i}) = \sum_{s_i' \in \mathcal{S}_i} \sum_{r_i \in \mathcal{R}_i} p(s_i', r_i \mid s_i, a_i, a_{-i}) \cdot [r_i + \gamma \cdot V_{\boldsymbol{\pi}}(s_i')] \,. \tag{2.45}$$

JAL reduces the variance encountered in value learning by directly controlling for the confounding effects of the other agents in the system. However, this method requires the assumption that opponent actions are visible, which is not always the case. This has led to the study of the *centralized learning and decentralized execution framework*. In this framework, a practitioner trains agents that can exploit additional information during training, so long as, the agent does not rely on the extra information during evaluation [Tan, 1993, Kraemer and Banerjee, 2016]. Researchers have investigated several forms of additional training information, such as opponent actions [Claus and Boutilier, 1998], opponent states [Rashid et al., 2018], and coordination signals [Greenwald and Hall, 2003].

A key question then becomes: how to create a policy that can be evaluated without relying on the additional information. This question has been primarily studied when assuming access to opponents' actions as extra training information. A popular approach is to decompose the joint-action value into independent Q-values for each agent [Guestrin et al., 2001, He et al., 2016, Sunehag et al., 2018, Rashid et al., 2018, Mahajan et al., 2019]. An alternative is to learn a centralized critic, which can train independent agent policies [Gupta et al., 2017, Lowe et al., 2017, Foerster et al., 2018b]. Some have proposed constructing metadata about the agent's current policies as a way to reduce the learning instability present in environments where the opponents' policies are changing [Foerster et al., 2017, Omidshafiei et al., 2017].

Most of the aforementioned techniques train the agents concurrently. In contrast, our context assumes that in each application of learning the opponent plays a distribution over a stationary, or non-learning, set of policies. This removes the need to account for moving targets within a learning operation [Hernandez-Leal et al., 2017], and rather relies on a broader iterative process to address the joint dynamics of multiagent learning [Foerster et al., 2018a, Tesauro, 2003].

2.2.2 Coplayer Generalization

Another major consideration that pertains to MARL is generalization across coplayers. There are three settings that are typically, but not exclusively, considered that I detail in this subsection.

Fixed Coplayers. In this first setting, the coplayers are assumed to always follow a fixed policy. Note, that this policy may be stochastic. This setting can most readily be treated as a vanilla RL problem. However, considering the coplayers explicitly can dramatically reduce sample complexity.

Learning Coplayers. In this setting, it is assumed that the coplayers are also learning over time. It is implicitly assumed that the coplayers are learning from the same shared experiences with the ego-centric learner. These shared experiences can serve as a coordination signal between learners. This can be useful for improving coordination across players [Tan, 1993]. And can competitively be exploited by recursively estimating the opponent's new policy and responding to it [Foerster et al., 2018a]. The majority of studies self-described as MARL are concerned with this setting.

Population-Based Coplayers. The final setting we will discuss is when coplayers sample a policy to play at the beginning of an episode. The set of eligible policies is called the "population" in MARL, and is analogous to a mixed strategy from GT [Lanctot et al., 2023]. The population is often partitioned into training and evaluation. The challenge with this setting is learning to respond to a potentially diverse set of coplayers. All the while, not "overfitting" to the training population such that you cannot coordinate, or get exploited, by the evaluation population. Population-based generalization analogously returns in our discussion of EGTA in Section 2.4.

Important to solutions to all settings is that any assumptions that are used when learning a policy must not need to be relied upon once learning is finished. For example, JAL assumes access to the coplayers' actions to do value learning. When playing a competitive game we cannot reasonably assume that our opponent will tell us their actions. Even in cooperative games, communication or observation may be limited in a way that we cannot definitively know all of our coplayers' actions. As a result, we want to ensure our learning algorithm produces a policy that carries-forward as few assumptions as possible into evaluation.

This framework for construction a learning algorithm has been referred to as *centralized training and decentralized execution*. Here, the centralization of each step refers to the fact that MARL algorithms often have a central coordinator querying all of the policies that can be used to exchange additional information. Therefore, in training, the learner may access privileged information regarding their coplayers due to the *centralized* nature of the learning algorithm. However, the final learned policy must not require said information, because it will be evaluated in a *decentralized* setting. In essence, this serves as rebranding of the train-test split used in supervised learning.

2.3 Game Theory

Game Theory (GT) supplies the notion of what it means to solve a game. GT is a branch of mathematics and economics that provides a framework for analyzing the strategic interactions between rational decision-makers. It analyzes situations in which the outcomes these decision-makers depends not only on their own actions but also on the actions of other decision-makers within the system.

2.3.1 Normal-Form Games

The foundation of GT lies in the concept of a game. Games represent strategic interactions through the players, strategies, and payoffs. The choice of representation is called the game's *form*.

A *normal form game* (NFG), describes a strategic interaction by way of a matrix where each cell corresponds to the players' payoffs for a choice of strategy per-player. Formally, a NFG is a three-tuple $\Gamma = (\mathbf{\Pi}, U, n)$. n is simply the number of players in the game. $\mathbf{\Pi}$ is the players' *strategy set*, where each player i's strategy is a distribution over policies[2] $\Pi_i = \{\pi_i^0, \dots, \pi_i^{k_i}\}$, where player i has k_i available policies (k_i may be infinite). U is the utility, payoff, matrix that for each selection of policies assigns all players' payoffs $U : \mathbf{\Pi} \rightarrow \mathbb{R}^n$.

		Player 1	
		Cooperate	Defect
Player 0	Cooperate	$(3,3)$	$(0,5)$
	Defect	$(5,0)$	$(1,1)$

Figure 2.2: **Prisoner's Dilemma.**

Figure 2.2 illustrates one such NFG called the Prisoner's Dilemma [Rapoport and Chammah, 1965, Tucker, 1950]. This dilemma presents each player with the opportunity to cooperate for

[2]These may also be called actions, decisions, etc. Policies are descriptions of behavior making analogies with RL readily apparent. Therefore, I will use policy throughout this dissertation.

mutual benefit, or betray their coplayer by defecting and achieving a higher personal reward. This game is played by 2 players $n = 2$. Each player has the same strategy set:

$$\Pi_0 = \Pi_1 = \{\text{Cooperate}, \text{Defect}\} = \{\pi^{\text{Cooperate}}, \pi^{\text{Defect}}\}.$$

And the payoff matrix U is shown in Figure 2.2. For example, if both players choose to cooperate, then they both receive a payoff of 3. However, if Player 0 chooses to defect, while Player 1 cooperates, then they respectively see payoffs of 5 and 0. When the players do not have meaningful names to distinguish between them, they will be referred to by their associated axis. In this example Player 0 would be the row player, and Player 1 the column (col) player.

At the start of a game each player selects their policy following a *strategy* $\sigma_i : \Pi_i \to [0, 1]$. A player is said to be playing a *pure strategy* if they deterministically select a policy. As pure-strategies are equivalent to their selected policy the strategy can be referred to as that policy directly $\pi_i \in \Pi_i$. Otherwise, the player is said to be playing a *mixed strategy* $\sigma \in \Sigma \equiv \Delta(\Pi_i)$, where Δ is the probability simplex of the player's strategy set Π_i. A strategy profile is an assignment of strategies to players. The player keeps their selected policy for the remainder of that gameplay, and only resamples it at the start of a new gameplay. In the case of NFGs, this may appear reductive, as the game only requires making a single decision. However, this distinction is important as we move onto our next game form (extensive).

2.3.2 Extensive-Form Games

Extensive-form games (EFG) offer an alternative game framework that encapsulates the temporal dynamics of the game [Kuhn, 1953]. The distinguishing feature of EFGs over NFGs is the explicit consideration of time. EFGs allow the additional specification of the ordering of individual player decisions and the information available to them for each decision. While a formal treatment of EFGs is not pertinent to this dissertation, I will use them here to introduce several concepts that help bridge GT and RL.

Figure 2.3 depicts an extensive-form game tree of the prisoner's dilemma we saw in Figure 2.2. The original prisoner's dilemma was treated as a *simultaneous* move game, where the players submit their decisions at the same time. In the extensive-form I have illustrated here, Player 0 makes the first decision at the hollow node. Their corresponding decision leads the game down its labelled branch. Then, Player 1, at the filled nodes makes their decision. Player 1 cannot distinguish between the situation where Player 0 had previously chosen cooperate or defect. As a result, the *information [state]* available to them is equivalent. The set of all nodes with the same information state, and are as a result indistinguishable, is called an *information set*. In our example's figure, the information set is shown with a dashed oval containing the equivalent information states. The

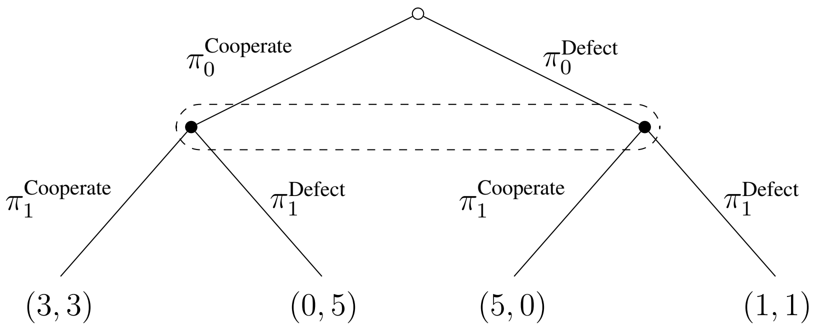


Figure 2.3: **Extensive-form prisoner's dilemma.**

leaves of the game tree (bottom points without circles) contain the payoff of all players if they collectively arrive at this end-point of the game.

2.3.3 Payoff Characteristics

When studying a game, an essential characteristic to consider is the players' payoff relationships. These characteristics can inform additional assumptions about respective game subclasses, significantly reducing the complexity required for game reasoning.

A notable category of games is *common-interest games*, where players share identical payoffs across all possible outcomes. This commonality can substantially simplify a game as rational players can be assumed to act cooperatively. Furthermore, cooperative games can be reinterpreted as a decentralized single-agent control problem where each player's action serves as a component of the cumulative "player" action [Oliehoek, 2012]. In all other games, the extent to which players' payoffs align dictates the level of self-interested behavior the respective player should adopt.

Anonymous games are games where the payoff from a chosen strategy depends only on the strategies employed by other players, not on who has adopted them. A subclass of anonymous games is *symmetric games* where the players additionally have the same utility function. This characteristic naturally reduces game complexity as various permutations of joint strategies are now represented by a single, representative combination.

Games can also be characterized by defining relationships between players' payoffs across strategy profiles. *General-sum games* represent the broadest case where no specific relationship is defined. On the other hand, *constant-sum games* are games where the combined payoff for all strategy profiles equals a fixed constant, c:

$$\forall \boldsymbol{\sigma} \in \boldsymbol{\Sigma}, \quad \sum_{i \in n} U_i(\boldsymbol{\sigma}) = c. \tag{2.46}$$

Constant-sum games can be additionally called *zero-sum games* by subtracting c from every payoff.

In these games, any gain in one player's payoff is a direct loss to another player's payoff. Zero-sum games present a significant assumption about the game's strategic nature, which can considerably reduce the complexity involved in solving and reasoning within these games, especially when compared to general-sum games.

2.3.4 Solution Concepts

A *solution* to a game is a strategy profile that instantiates a *solution concept*. A solution concept defines a set of rules for predicting solution(s) to a game. The choice of solution concept is the choice of which solutions you find meaningful for analysis.

A common choice for solution concept is the *Nash equilibrium (NE)*. These equilibria describe solutions where no one player can receive a higher payoff by unilaterally changing their strategy. Formally, a joint strategy $\boldsymbol{\sigma}^*$ is said to be a Nash equilibrium if:

$$\forall i \in n, \quad \forall \sigma_i \in \Sigma_i, \quad U_i(\sigma_i^*, \sigma_{-i}^*) \geq U_i(\sigma_i, \sigma_{-i}^*).$$

Throughout this dissertation I empirically demonstrate the performance of game-solving algorithms with respect to NE. However, the algorithms are flexible to their choice of solution concept.

Related to the study of solutions is the study of *best responses* (BR). BR policies are those policies that receive the highest payoff against a fixed coplayer strategy σ_{-i}:

$$\pi_i^* \in \text{BR}(\sigma_{-i}) \qquad \text{iff} \qquad \forall \pi_i \in \Pi_i, \quad U_i(\pi_i^*, \sigma_{-i}) \geq U_i(\pi_i, \sigma_{-i}). \tag{2.47}$$

2.3.5 Evaluation Methods

Assessing the quality of a strategy within a game poses a challenge. A typical measure for quality is the payoff a strategy yields. However, it is unrealistic to assume that we can anticipate our coplayers' strategies before the game is played. This creates a dilemma as our payoff will fluctuate based on any changes in our coplayers' strategies.

An alternative solution is to evaluate a strategy using a hindsight-based approach. In this case, a high-quality strategy is one that minimizes *regret* relative to a coplayer's strategy. Regret represents the potential gain a player could have achieved in hindsight by deviating from their chosen strategy. Formally, the regret of player i towards the joint-strategy $\boldsymbol{\sigma}$ can be quantified as:

$$\text{Regret}_i(\boldsymbol{\sigma}) = \max_{\pi_i \in \Pi_i} U_i(\pi_i, \sigma_{-i}) - U_i(\boldsymbol{\sigma}). \tag{2.48}$$

Using the concept of regret, we can define and measure the stability of a strategy profile. A solution is considered stable if no player can increase their payoff by switching to a different strategy. This can be measured by totaling the regret of each player:

$$\text{SumRegret}(\boldsymbol{\sigma}) = \sum_{i \in n} \max_{\pi_i \in \Pi_i} U_i(\pi_i, \sigma_{-i}) - U_i(\boldsymbol{\sigma}). \tag{2.49}$$

This metric is occasionally referred to as NashConv, as it can be interpreted as a measure of the distance from a Nash equilibrium. However, this metric is applicable to various solution concepts, so I refer to it as SumRegret in this dissertation.

2.4 Empirical Game Theoretic Analysis

Algorithms for computing solutions in GT typically require complete descriptions of the game. This can prove limiting on games with real-world complexity. On the easy end of the problems is that their are hundreds of policies or players, and as a result, the runtime complexity for solving the game is untenable. On the harder end, there may be an infinite number possible policies that cannot be written down let alone computed over.

Empirical Game Theoretic Analysis (EGTA) addresses this problem by reasoning over approximate game models, called *empirical games*, estimated by simulation over a restricted strategy set. I will denote empirical games using the same notation as non-empirical games, but with the addition of a hat $\hat{}$. For example, an *empirical normal-form game* (ENFG) is then denoted as $\hat{\Gamma} = (\hat{\boldsymbol{\Pi}}, \hat{U}, n)$. Where $\hat{\boldsymbol{\Pi}} \subseteq \boldsymbol{\Pi}$ is a restricted strategy set, and $\hat{U} \approx U$ is an estimated payoff function.

Walsh et al. [2002] first demonstrated the efficacy of EGTA in a study of pricing and bidding games. Bear in mind, that the fidelity of the empirical game is primarily impacted by the the size of the restricted strategy set, and the choice of which strategies are included. The general question of which strategies to include in an empirical game was framed by Jordan et al. [2010] as the *strategy exploration problem*. Phelps et al. [2006] introduced the idea of automatically extending the restricted strategy set through optimization. They showed that this was possible by applying a genetic search algorithm over policy space. Schvartzman and Wellman [2009b] proposed to use RL to derive *approximate best-responses* (ABR) to the current empirical game's Nash equilibrium. The PSRO algorithm generalized this approach to include DRL and any solution concept of the empirical game [Lanctot et al., 2017].

2.4.1 Policies as Actions

A subtle detail that I employed in Section 2.3 is to treat policies as atomic in NFGs. This approach provides a computational efficiency advantage for game reasoning, as it circumvents the necessity of reasoning over a complete game tree, where each decision point exponentially escalates the complexity. However, it is important to note that while they are treated as atomic for NFG reasoning, they are not treated atomically for BR reasoning using RL. BR through RL, contrastingly, evaluates each action decision in the game.

While the policy abstraction affords convenience, as with any choices in modeling, it can come with downsides. The amount of time that is abstracted by a policy is directly reflected in the combinatorial growth of the game matrix. This elevates the importance of a conservative selection which policies to include in the empirical game. A liberal inclusion may leave the empirical game's complexity in its original and intractable state.

2.4.2 Policy-Space Response Oracles

Policy-Space Response Oracles (PSRO) is a general learning-based game-solving algorithm that interleaves DRL and EGTA [Lanctot et al., 2017]. As its namesake suggests, it takes inspiration from the Double Oracle (DO) [McMahan et al., 2003] algorithm. In DO, a game is solved by iteratively, across players, including a BR to the current NE. PSRO operates similar to DO by alternating between game-reasoning and strategy exploration.

PSRO begins with either player i having a provided initial strategy set $\hat{\Pi}_i^0$, or the initial strategy set being initialized to contain a policy that plays randomly $\hat{\Pi}_i^0 = \{\pi_i^{\text{random}}\}$. The payoffs of each strategy profile are then estimated through play, typically completed through computer simulation. These payoffs and policies constitute PSRO's initial empirical game $\hat{\Gamma}^0$. PSRO then proceeds to iterate between game-reasoning and strategy exploration, until it has captured an approximate solution to the real game.

Game reasoning involves solving the current empirical game. This requires specification of a solution concept. The abstract subroutine that computes the current solution is called the *Meta-Strategy Solver (MSS)*, and is functionally defined as follows:

$$\text{MSS} : \hat{\Gamma} \to \boldsymbol{\sigma}^{*,e}, \tag{2.50}$$

the resultant strategy $\boldsymbol{\sigma}^{*,e}$ is distinguished as both a solution with $*$ and the current iteration, "epoch", of empirical game solving with e. The MSS serves the dual purpose of checking for convergence of PSRO and a solver. An example choice of a MSS is the NE, as done in DO, which could be operationalized through any number of NE-solving algorithms.

If the empirical game has not converged, the algorithm proceeds to enrich the empirical game through strategy exploration. PSRO strategy exploration method is to include ABR, computed using DRL, to the empirical game's current solution:

$$\pi_i^e \in \text{ABR}(\sigma_{-i}^{*,e-1}).$$

The new policies are then included into the empirical game, and the resulting new strategy profiles are estimated. It is within PRSO's strategy exploration method that we can understand the algorithm's opaque name. Oracle methods are those that assumed to freely and instantly provide answers, as if from an all-knowing oracle. In the case of DO, as the game was non-empirical, the oracle simply looked-up the BR in the game's full payoff matrix. Therefore, response oracles, are oracles that produce responses. And finally, the modifier, policy-space, reflects the absence of a true oracle, and that instead an approximate oracle must be employed that searches over policy space. In this case, the search is performed using DRL.

Algorithm 1: Policy-Space Response Oracles [Lanctot et al., 2017]

```
Input: Initial policy sets for all players $\boldsymbol{\Pi}^0$
Simulate utilities $\hat{U}^{\boldsymbol{\Pi}^0}$ for each joint $\boldsymbol{\pi}^0 \in \boldsymbol{\Pi}^0$
Initialize solution $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$
while epoch e in $\{1, 2, \dots\}$ do
    for player $i \in n$ do
        for many episodes do
            $\pi_{-i} \sim \sigma_{-i}^{*,e-1}$
            Train $\pi_i^e$ over $\tau \sim (\pi_i^e, \pi_{-i})$
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$
    Simulate missing entries in $\hat{U}^{\boldsymbol{\Pi}^e}$ from $\boldsymbol{\Pi}^e$
    Compute a solution $\sigma^{*,e}$ from $\hat{\Gamma}^e$
Output: Current solution $\sigma_i^{*,e}$ for player $i$
```

These two routines are iteratively applied into convergence. Convergence is typically determined by when no further ABR can be computed for any players. Practically, convergence can also be applied by a limited walltime on the algorithm. Algorithm 1 contains PSRO's pseudocode.

The PSRO framework was designed to provide a flexible template for reasoning about complex games exploiting DRL as a powerful BR technique. A key idea of PSRO is to abstract the response target beyond NE, through the concept of a Meta-Strategy Solver (MSS). MSSs offer a lens that unifies many existing algorithms in multiagent learning and game theory. With an NE solver as its MSS, PSRO corresponds to DO. Different algorithms for computing NE (e.g., based on linear-programming, replicator dynamics [Taylor and Jonker, 1978], regret-minimization [Blum and Mansour, 2007], or regret-matching [Hart and Mas-Colell, 2000] can be considered distinct MSSs, to the extent they may produce results differing in accuracy or equilibrium selection. An MSS that selects a uniform distribution over policies in the current strategy set produces the classic technique of fictitious play [Brown, 1951]. Self-play [Silver et al., 2016, Heinrich, 2017] results from the MSS that returns a pure strategy containing only the newest policy. Table 2.1 compares a selection of these algorithms by their MSSs output.

Table 2.1: **Overview of previous MARL algorithms as versions of PSRO differing in MSS.** The lists under MSS correspond to the solution returned for each agent with a coefficient added for each currently learning oracle as the final item in the list. e corresponds to the current epoch of PSRO.

Algorithm	Meta-Strategy Solver
Independent Reinforcement Learning	$[0, 0, \ldots, 0, 1]$
Iterative Best-Response [Monderer and Shapley, 1996]	$[0, 0, \ldots, 1, 0]$
Fictitious Play [Brown, 1951]	$[1/e, 1/e, \ldots, 1/e, 0]$
Fictious Self-Play [Heinrich et al., 2015]	$[\frac{1}{e+1}, \frac{1}{e+1}, \ldots, \frac{1}{e+1}, \frac{1}{e+1}]$

Recent work has investigated and evaluated a variety of new MSSs for strategy evaluation. For example, Wang et al. [2019] propose a weighted combination of NE and uniform profiles. Wright et al. [2019] use NE as primary MSS, but then adjust the BR to improve response to a decay-weighted linear combination of previous solutions. Omidshafiei et al. [2019] employ Markov-Conley Chains to define a solution concept that relates to their policy evaluation measure α-rank, and Muller et al. [2020] investigated its use as an MSS. Marris et al. [2021] proposed MSSs based on various forms of correlated equilibrium. Indeed, any established or new solution concepts is a ready candidate to serve as an MSS for strategy exploration.

2.4.3 Regret in Empirical Games

The regret of a solution in an empirical game may not exactly match regret in the real game. We can see this by comparing empirical-game regret with the real game regret (Equation 2.48):

$$\text{Regret}_i(\boldsymbol{\sigma}) = \max_{\pi_i \in \hat{\boldsymbol{\Pi}}_i} U_i(\pi_i, \sigma_{-i}) - U_i(\boldsymbol{\sigma}). \tag{2.51}$$

The crux of the differences lies in the *[deviation] set* of policies considered for deviation. In the real game, the full policy space $\boldsymbol{\Pi}_i$ is considered. Whereas, in this measure of regret, only the policies contained with the empirical game are available $\hat{\boldsymbol{\Pi}}$. Policies absent from the restricted strategy-set $\hat{\boldsymbol{\Pi}}$ present opportunities for miscalculations of regret.

When comparing or evaluating game-solving algorithms, additional estimates of regret may be employed. To understand this, let us first reformulate regret to be conditional on the deviation set:

$$\text{Regret}_i(\boldsymbol{\sigma} \mid \boldsymbol{\Pi}) = \max_{\pi_i \in \boldsymbol{\Pi}} U_i(\pi_i, \sigma_{-i}) - U_i(\boldsymbol{\sigma}). \tag{2.52}$$

One avenue for improving the estimate of regret is by including a set of known evaluation policies $\boldsymbol{\Pi}^{\text{eval}}$. These could be any of previous benchmark policies, heuristic policies, or even hand-designed policies. Evaluation policies can also serve as a common baseline to estimate the relative regret of algorithms:

$$\text{Regret}_i(\boldsymbol{\sigma} \mid \hat{\boldsymbol{\Pi}} \cup \boldsymbol{\Pi}^{\text{eval}}) \qquad \text{or} \qquad \text{Regret}_i(\boldsymbol{\sigma} \mid \boldsymbol{\Pi}^{\text{eval}}). \tag{2.53}$$

If you are comparing multiple game-solving algorithms, you can considered the *combined game* constructed by joining each method's empirical game:

$$\text{Regret}_i \left(\boldsymbol{\sigma} \middle| \bigcup_{\text{method}} \hat{\boldsymbol{\Pi}}^{\text{method}} \right). \tag{2.54}$$

This method takes advantage of every available estimate of regret within a comparison. This version of regret can be costly to compute, because it requires estimate the payoff of all the new strategy profiles. Rendering it intractable for comparisons with many methods, or where each empirical game is already sizable.

2.5 Transfer Learning

Transfer learning is the study of reusing knowledge gained in one context to facilitate learning in a related but different context. Opportunities for transfer may arise in learning tasks, domains, policies, or any other learning target. Within the field of transfer learning, this study addresses two main questions: what type of knowledge is transferred, and how the knowledge is transferred. Both questions are framed within the context of a game, where the knowledge consists of response policies, and the transfer target is a different strategic scenario.

Previous work on how to transfer knowledge has tended to follow one of two main directions [Pan and Yang, 2010, Lampinen and Ganguli, 2019]. The *representation transfer* direction considers how to abstract away general characteristics about the task that are likely to apply to later problems. Ammar et al. [2015] present an algorithm where an agent collects a shared general set of knowledge that can be used for each particular task. The second direction directly transfers parameters across tasks; appropriately called *parameter transfer*. Taylor et al. [2005] show how policies can be reused by creating a projection across different tasks' state and action spaces.

In the literature, transferring knowledge about the opponent's strategy is considered intra-agent transfer [Silva and Costa, 2019]. The focus of this area is on *adapting to other agents*. One line of work in this area focuses on ad hoc teamwork, where an agent must learn to quickly interact with new teammates [Barrett and Stone, 2015, Bard et al., 2020]. The main approach relies on already having a set of policies available, and learning to select which policy will work best with the new team [Barrett and Stone, 2015]. Banerjee and Stone [2007] propose learning features that are independent of the game, which can either be qualities general to all games or strategies.

In contrast to prior work, our focus is not on adapting to entirely new opponents, but rather on transferring knowledge about response policies to new configurations or distributions of already encountered opponent policies. In other words, responses to opponent policies are the *source* of information to transfer.

2.5.1 Multi-Task Learning

Multiagent learning is analogous to multi-task learning. In this analogy, responding to each strategic context is analogous to solving a different task. The opponents' strategies relate to distributions over shared sets of tasks. Similar analogies from strategies to tasks can be made with objectives, goals, contexts, etc. [Kaelbling, 1993, Ruder, 2017].

The multi-task learning community has broadly categorized learnable knowledge into two groups [Snel and Whiteson, 2014]. *Task-relevant* knowledge pertains to a specific task [Jong and Stone, 2005, Walsh et al., 2006], while *domain-relevant* knowledge is common across all tasks [Caruana, 1997, Foster and Dayan, 2002, Konidaris and Barto, 2006]. Some work has bridged the gap between these settings; for example, knowledge about a task could be a curriculum to apply across tasks [Czarnecki et al., 2018]. In task-relevant learning, a leading method is to identify state information that is irrelevant to decision making and abstract it away [Jong and Stone, 2005, Walsh et al., 2006]. Our work falls into the same task-relevant category, where we are interested in learning responses to specific opponent policies.

2.6 Studied Games

In this section I introduce the more complex games that are studied in this dissertation.

2.6.1 Running With Scissors

Running With Scissors (RWS) is a temporarily extended version of Rock Paper Scissors (RPS) [Leibo et al., 2021]. The agents begin by collecting rock, paper, and scissor items scattered throughout the gird-world. These are added to the player's inventory v_i, which is initialized to have one of each item. The game ends when a player challenges the other to play RPS. Each player plays a distribution over the actions following the distribution of items in their inventory. The reward can then be calculated as:

$$r_i = \frac{v_i}{\|v_i\|} M \left(\frac{v_{-i}}{\|v_{-i}\|} \right)^{\mathrm{T}} = -r_{-i}, \qquad M = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}. \tag{2.55}$$

The game is a two-player zero-sum game with a small state and action space, enabling inexpensive simulation. The general map layout it depicted in Figure 2.4. In it we can see that the players randomly spawn in a fixed set of places. Moreover, items within the grid world can spawn both deterministically and stochastically. This means that if an agent does not know what spawned in a particular spot they cannot accurately infer the opponent's inventory. This is critical, because the game is partially observable. Agents are only able to view a small 5×5 sub-grid in around their position instead of the full 13×21 grid.

A particular instance of gameplay is provided in Figure 2.5. In this game, we can see that the starting observations of each player can view the spawn of two randomly spawned items (shown in the two right sub-grids). This gives each player private information of the state of the game.

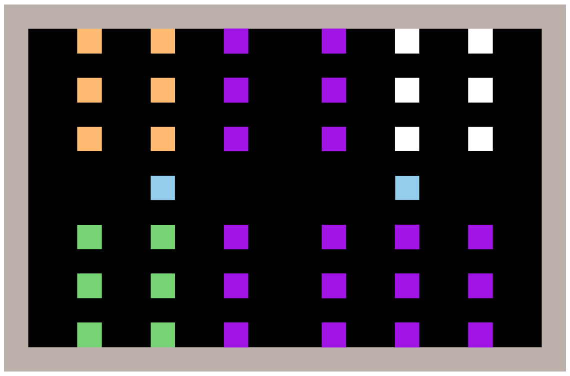


Figure 2.4: **RWS map layout.** The blue squares represent the possible spawn points of the players. Items either spawn deterministically as rock (orange), paper (white), scissors (green); or, one of the three possible items is randomly spawned into each position (purple). Black cells are empty, and light-gray represents walls.

2.6.2 Gathering (aka Harvest)

In Gathering, players move around an orchard picking apples. The challenging commons element is that an apple's regrowth rate is proportional to the number of nearby apples, so that socially optimum behavior would entail managed harvesting. Self-interested agents capture only part of the benefit of optimal growth, thus non-cooperative equilibria tend to exhibit collective overharvesting. The game has established roots in human-behavioral studies [Janssen et al., 2010] and

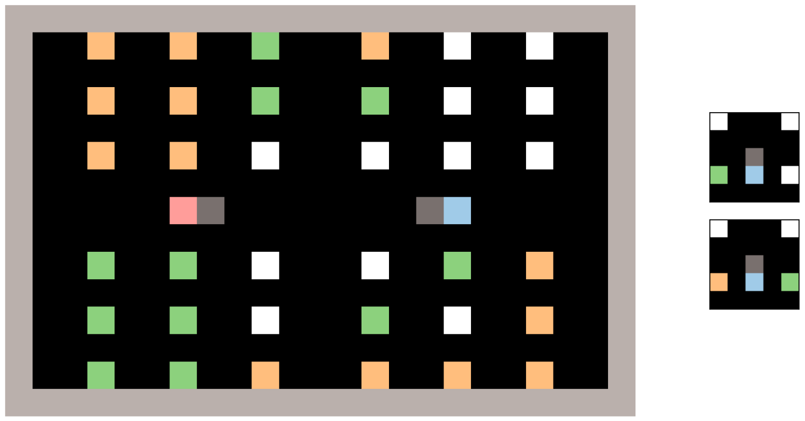


Figure 2.5: **RWS example observations.** (Left) The full state of the RWS game. (Right) The two player's observations. Blue is used to represent self in both observations, whereas red is used in the full state to distinguish between the two agents.

in agent-based modeling of emergent behavior [Pérolat et al., 2017, Leibo et al., 2017, 2021].

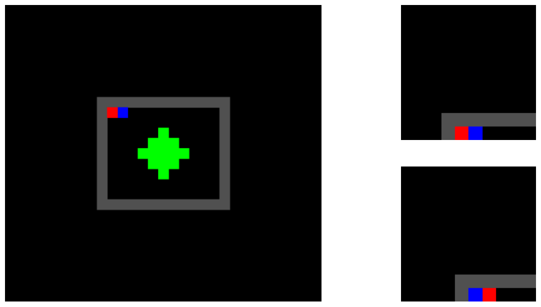


Figure 2.6: **Gathering game with categorical observations and the small map.** (Left) The full state of the game. (Right) The player observations.

Another benefit of the Gathering game is how it can be easily customized to study a variety of game qualities. In this dissertation I consider variants of the game that differ in ways listed below:

- The map specifying the player spawn points, apple locations, and walls.
- The observations either being categorical, or RGB images.
- The number of players.

Many qualities of the game are common across all settings. At each step of the game each player receives a partial observation of the game. The observation is a small rectangular region in front of the agent referred to as its viewbox. Actions are then simultaneously taken and privately selected for each player. The possible actions include moving in the four cardinal directions, rotating either way, tagging, or no action. A successful tag temporarily removes the other player from the game, but can only be done to other nearby players. Players earn a reward of 1 if their actions result in them landing on an apple, thereby picking it up.

For our initial experiments, I use a symmetric two-player version of the game, where in-game entities are represented categorically [HumanCompatibleAI, 2019]. This categorical representation facilitates faster experimentation and simplifies the interpretation of results. Figure 2.6 depicts the game state and player observations. Each player has a 10×10 viewbox within their field of vision. The cells of the grid world can be occupied by either agent shown in red and blue, the apples shown in green, or a wall in gray. I also use categorical observations for a three-player version of the game on a map called "open", which is shown in Figure 2.7.

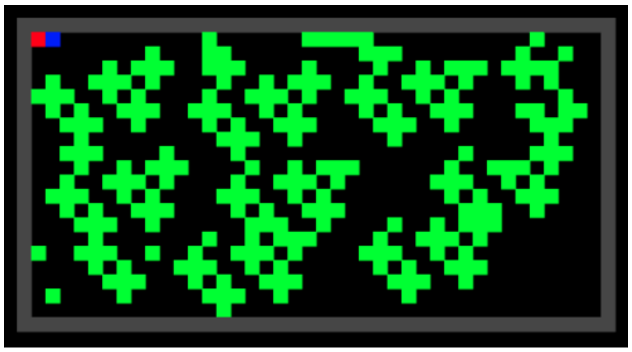


Figure 2.7: **Gathering game with categorical observations and the open map.**

Another variant of this commons game that I use features RGB observations. To further distinguish this game, I refer to it as Harvest: RGB. Harvest: RGB is exactly the harvest implementation from MeltingPot [Leibo et al., 2021] with the same orchard map. A rendering of the game state and observations is shown in Figure 2.8. The main difference between the Harvest versions is that the observations are $88 \times 88 \times 3$ images of the 11×11 viewbox in front of them. There are also minor differences in the implementation of tagging and apple respawn mechanism.

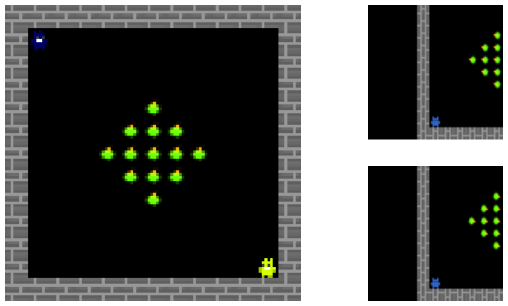


Figure 2.8: **Harvest: RGB game.** (Left) The full state of the game. (Right) The player observations.

Part II

Strategic Knowledge Transfer

CHAPTER 3

Opponent Mixture Transfer Problem

Strategic knowledge is dependent on the specific strategic context defined by the coplayers' strategies. This type of knowledge naturally comes up in many challenges of multiagent systems, such as coordination, establishment of conventions, and communication.

In the realm of our market example, strategic knowledge guides your interactions with vendors and other market participants. It equips you with insights, such as the realization that the market gets crowded after 10:30 and parking becomes difficult to find. This knowledge further assists you in bargaining with a vendors who, for instance, may need to dispose of their Honeycrisp apples before they spoil And at a more abstract level, strategic knowledge relates to knowing which language to use and the sociolinguistic conventions of the area.

In this part of the dissertation I focus on strategic knowledge and explore the problem of its transfer. As we defined in Section 1.3, strategic knowledge refers to predictions that depend on the strategies employed by an agent's coplayers. This definition is quite broad, as often an infinite space of predictions are possible that depend on the coplayers' strategies. Instead, with the overall goal of game solving in mind, it makes sense to focus on strategic knowledge that is effective against the given coplayers' strategies. Computation of a response policy is by definition relative to opponent strategies, and thus incorporates what we are referring to as performant strategic knowledge. For this reason, I adopt a working definition of strategic knowledge which focuses on the best-response predictions, contingent on the coplayers' strategies. This covers any artifacts that contribute to the predictions such as the accumulated experience, computed value function(s), etc. As a more flexible approach, I will also regard approximate best-response policies or response policies as forms of strategic knowledge.

Having defined strategic knowledge, we now shift our focus to the concept of its transfer.

Strategic knowledge transfer involves moving knowledge from a specific, or *source*, strategic context to enable more effective play in a different, or *target*, strategic context. For effective transfer to occur, it is crucial that the source and target contexts share common features. Consequently, a well-structured problem in strategic knowledge transfer demands that some behaviors of the coplayers be consistent across contexts. Within the scope of game solving, strategic knowledge transfer is clearly defined by exploiting the consistency and expansion of the coplayer's strategy set across different iterations. Thus, with every iteration, the previous iteration acts as the source context and the current iteration becomes the target context.

The opportunity here lies in avoiding redundant learning when playing against a largely known group of coplayers. In other words, we aim to prevent relearning against coplayer policies that have already been considered. This setup frames a special case of the one-step transfer problem, Problem 2, where the coplayer's strategy set remains constant across different transfer contexts.

With the definitions above, we are ready to give a precise definition of a particular problem in strategic knowledge transfer. Namely, suppose we have a set of response policies, each an ABR to a specified opponent policy. Now, we are faced with a randomized opponent, whose behavior is a distribution, or mixture, over these same opponent policies. Intuitively, the response policies should contain information useful for generating an ABR to this mixture.

This setup frames what we call the *opponent mixture transfer problem*. At a high level, this question asks how response policies to all of the opponent's pure strategies, or policies, can be transferred to generate a response to a given opponent mixed strategy.

Problem 3 (Opponent Mixture Transfer Problem)**.** *Given:*

- *the mixed strategy played by the opponent,* $\sigma_{-i} \in \Delta(\Pi_{-i})$*, and*
- *the set of responses to each opponent policy,* $\{\mathrm{BR}(\pi_{-i})\}_{\pi_{-i} \in support(\sigma_{-i})}$,

construct a response, $\mathrm{BR}(\sigma_{-i})$*, to the opponent mixture* σ_{-i}*.*

The efficacy of a transfer method is judged relative to the cost of deriving a response policy without strategic transfer—that is, by explicit training against σ_{-i}.

The opponent mixture transfer problem may arise in various contexts. Our particular motivating context is within population-based game-solving algorithms, which compute response policies to play against mixtures drawn from an evolving population of policies. The transfer opportunity arises from the persistence of components of the opponent population across response computations. Thus, BRs calculated for opponent policies present in the population are likely to remain relevant for a significant period as the population evolves.

I first consider the direct transfer of policies (Chapter 4) that model probabilities over actions. Observing practical limitations in this approach, we turn our investigation to value-based policies

(Chapter 5). Value-based policies directly model the expected discounted return that an agent should expect for taking an action. I introduce a method, *Q-Mixing* [Smith et al., 2023a], that approximately constructs a best-response to a mixed strategy by appropriately averaging the values of each opponent response policy. Critical to the success of Q-Mixing is the ability of the resulting agent to maintain an accurate belief in the current opponent policy. In Chapter 6, we explore belief maintenance methods and the factors contributing to their success.

I next introduce two methods for strategic knowledge transfer to reduce the computational cost of PSRO (Chapter 7) [Smith et al., 2021]. *Mixed-Oracles* learns separate response policies to each policy in the opponent's mixed strategy; and then, combines the response policies to approximate a response to the full mixed strategy. *Mixed-Opponents* constructs a novel opponent policy that represents an amalgamation of the full mixed strategy, facilitating the transfer of non-optimal behaviors. A response is then calculated against this novel policy, benefiting from the variance reduction resulting from the elimination of sampling over opponent policies. Both of these algorithms employ, but are not limited to, Q-Mixing as a subroutine capable of aggregating strategies into a single policy. When compared to standard PSRO, both methods exhibit a decrease in the cumulative cost necessary to solve a game.

CHAPTER 4

Direct Policy Transfer

Ideally, we want a method that solves the opponent mixture transfer problem by a direct operation on the component response policies. By *direct operation*, I mean a solution that does not require any additional assumptions about the response policies' implementations or underlying derivations, thereby making it applicable to the broadest possible range of contexts. Effective play would be transferred from the knowledge contained in the response policies, weighted by probabilities that their targets are played by the opponent.

A general solution of this kind is not possible, however, as we demonstrate below through a counterexample. The counterexample shows, at minimum, that all actions necessary for the mixed-strategy response must be among the actions possibly taken in the pure-strategy responses. As a result, a direct operation is not possible, because additional assumptions must be taken. To understand this limitation, we introduce an example we call the *direct policy transfer counterexample game*, depicted in Figure 4.1.

		Player 2	
		π_2^0	π_2^1
	π_1^0	10, -10	-10, 10
Player 1	π_1^1	-10, 10	10, -10
	π_1^2	1, -1	1, -1

Figure 4.1: **Direct policy transfer counterexample game.**

Let us take the perspective of Player 1 in this game, and suppose we are tasked with responding to the uniform mixed strategy of Player 2. Following the opponent mixture transfer problem, Problem 3, our inputs are the opponent's mixture $\sigma_2^{\mathrm{U}} \leftarrow (0.5, 0.5)$, and the BRs to each of their supported policies:

$$\mathrm{BR}(\pi_2^0) \rightarrow \pi_1^0 \qquad\qquad \mathrm{BR}(\pi_2^1) \rightarrow \pi_1^1.$$

From these inputs alone, however we cannot construct the correct BR(σ_2^{U}), which is π_1^2. In this case, the BR to the mixture does not involve the response policies to mixture components. We would need additional information to identify π_1^2 as the optimal response policy.

Despite this counterexample, there may still be games where direct policy transfer can be effective. In some cases relaxations may also be safely made to guarantee correctness. Therefore, for completeness, we now discuss methods for direct policy transfer that may be used in these settings.

The resultant transferred policy must define the same state distribution over the game as BR(σ_{-i}). This property is called realization equivalence and was introduced to related stochastic policies (i.e., behavioral strategies) and mixed strategies in sequence-form games [von Stengel, 1996, Koller and Megiddo, 1992]. Establishing realization equivalence between our transferred policy and a target BR(σ_{-i}) enables us to verify successful transfer algorithms. When this is the case, the set of response policies (to each opponent policy) can be purified[1] into a single policy representing a response to a mixed strategy opponent.

In order to formally define visitation frequencies we must first establish some primitives. Let $d^0 : \mathcal{S} \to [0,1]$ or $d^0 \in \Delta(\mathcal{S})$ by the initial state distribution. From this we can derive $\mathbb{P}^\pi$ as the probability that the random variables, denoted by capital script, take on the assigned values when π is acting in the environment. This allows us to describe many useful probabilities:

$$\mathbb{P}^\pi(S_0 = s) = d^0(s)$$
$$\mathbb{P}^\pi(A_t = a \mid S_t = s) = \pi(a \mid s)$$
$$\mathbb{P}^\pi(S_t = s_t, A_t = a_t \mid S_0 = s_0) = \pi(a_t \mid s_t)\mathbb{P}^\pi(S_t = s_t \mid S_0 = s_0)$$

$$\mathbb{P}^\pi(S_t = s_t \mid S_0 = s) = \sum_{s_{t-1}} \sum_{a_{t-1}} p(s_t \mid s_{t-1}, a_{t-1})\pi(a_{t-1} \mid s_{t-1})\mathbb{P}^\pi(S_{t-1} = s_{t-1} \mid S_0 = s_0)$$

It is worth noting explicitly here that this notation takes the viewpoint of one agent; where, they view all other agents as part of the environment. From these tools we formally define visitation frequencies in Definition 1.

Definition 1 (Visitation Frequency [Puterman, 1994, §6.9.2])**.** *A visitation frequency is the discounted probability of a policy π occupying either a state $\rho^\pi : \mathcal{S} \to [0,1]$ or joint state-action $\rho_\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$. For the initial state distribution d^0 and the probability of reaching state and/or action s, a after t timesteps and starting in state s^0 as* $\Pr^\pi\left(S^t = s^t, A^t = a^t \mid S^0 = s^0\right)$

[1]The moniker of purification refers to representing a set of policies as a *pure*-strategy (policy).

with discount factor γ the visitation frequency is as follows:

$$\rho^\pi(s,a) \doteq \sum_{s^0 \in \mathcal{S}} d^0(s^0) \sum_{t=0}^{\infty} \gamma^n \cdot \mathbf{Pr}^\pi \left(S^t = s, A^t = a \mid S^0 = s^0 \right), \tag{4.1}$$
$$\rho^\pi(s) \doteq \sum_{a \in \mathcal{A}(s)} \rho^\pi(s,a).$$

This quantity is also referred to as the occupancy frequency *or* occupancy *of a policy.*

Through visitation frequencies we can establish a method for purifying a mixed strategy. This is accomplished by weighting the policies by the likelihood of their respective play, within the mixed strategy, in a particular state. This relationship is established below in Theorem 2.

Theorem 2. *The purified policy π^σ of a mixed strategy σ is a the convex combination of each policy's action-distribution and the likelihood of that policy for each state:*

$$\forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}(s), \quad \pi^\sigma(a \mid s) = \sum_{\pi \in support(\sigma)} p(\pi \mid s, \sigma) \cdot \pi(a \mid s). \tag{4.2}$$

Assuming all state-action pairs are reachable by the pure strategies $\pi \in support(\sigma)$.

Proof. We begin by establishing the high-level relationship between the purified-policy and its visitation frequency $x^\sigma(s,a)$ via simple probability rules:

$$x^\sigma(s,a) = p(s \mid \sigma) \cdot \pi^\sigma(a \mid s), \tag{4.3}$$

$$\pi^\sigma(a \mid s) = \frac{x^\sigma(s,a)}{p(s \mid \sigma)}. \tag{4.4}$$

The remainder of this proof focuses on removing $p(s \mid \sigma)$ from Equation 4.4, by unpacking $x^\sigma(s,a)$ to contribute a corresponding $p(s \mid \sigma)$ for reduction. To this end, we first establish the policy-likelihood for later use:

$$\begin{aligned} p(\pi \mid s, \sigma) &= \frac{p(s \mid \pi) \cdot p(\pi \mid \sigma)}{p(s \mid \sigma)} \\ &= \frac{p(s \mid \pi) \cdot \sigma(\pi)}{p(s \mid \sigma)} \\ p(\pi \mid s, \sigma) \cdot p(s \mid \sigma) &= p(s \mid \pi) \cdot \sigma(\pi) \end{aligned} \tag{4.5}$$

Now, we focus on expanding $x^\sigma(s,a)$ to include $p(s \mid \sigma)$ using Equation 4.5:

$$\begin{aligned}
x^\sigma(s,a) &= \sum_\pi p(\pi, s) \cdot \pi(a \mid s) \\
&= \sum_\pi p(s \mid \pi) \cdot p(\pi) \cdot \pi(a \mid s) \\
&= \sum_\pi p(s \mid \pi) \cdot \sigma(\pi) \cdot \pi(a \mid s) \\
&= \sum_\pi p(\pi \mid s, \sigma) \cdot p(s \mid \sigma) \cdot \pi(a \mid s) \\
&= p(s \mid \sigma) \cdot \sum_\pi p(\pi \mid s, \sigma) \cdot \pi(a \mid s) \qquad (4.6)
\end{aligned}$$

Finally, we substitute Equation 4.6 into Equation 4.4:

$$\begin{aligned}
\pi^\sigma(a \mid s) &= \frac{x^\sigma(s,a)}{p(s \mid \sigma)} \\
&= \frac{p(s \mid \sigma) \cdot \sum_\pi p(\pi \mid s, \sigma) \cdot \pi(a \mid s)}{p(s \mid \sigma)} \\
&= \sum_\pi p(\pi \mid s, \sigma) \cdot \pi(a \mid s)
\end{aligned}$$

□

The construction of a mixed strategy best-response policy is a direct application of this operation. The response policies to each respective opponent policies are played in the same distribution that the opponent plays.

There are two limitations with this approach (1) the need for a policy-state likelihood, and (2) that state-action pairs must be reachable by the component policies. Computing a policy-state likelihood requires a $\mathcal{O}(\mathcal{S}\mathcal{A}\Pi)$ computation. This cost rivals directly computing a response to the mixed strategy directly, and thus renders the overall approach unavailing. The other limitation of policy-based transfer approaches is that we have assumed that all joint state-actions are reachable. Unfortunately, this assumption is difficult to guarantee outside of simple games. Due to these outstanding issues, we close the door on policy-based transfer methods and leave them for future work; instead turning towards value-based transfer methods.

CHAPTER 5

Value Function Transfer

Quality... you know what it is, yet you don't know what it is. But that's self-contradictory. But some things are better than others, that is, they have more quality. But when you try to say what the quality is, apart from the things that have it, it all goes poof! There's nothing to talk about. But if you can't say what Quality is, how do you know what it is, or how do you know that it even exists? If no one knows what it is, then for all practical purposes it doesn't exist at all. But for all practical purposes it really does exist.

— Robert M. Pirsig,
Zen and the Art of Motorcycle Maintenance

We now turn towards adopting assumptions about the underlying implementation of the response policies. Specifically, we consider *value-based* policies, which are policies derived from an action-value function. As value-based policy derivation is a ubiquitous technique in reinforcement learning, it lends support to the pursuit of exploiting value representations in strategic transfer.

With respect to the opponent mixture transfer problem, assuming value-based policies means that each response policy BR(π_{-i}) has an associated value function $Q_i(\cdot \mid \pi_{-i})$. The value function captures the expected return of *all state-action pairs*, which allows the evaluation of all actions relative to the Q-value function, as opposed to just the actions in the support of its policy.

The distinction is clear when we recall direct policy transfer counterexample game from Chapter 4. We could not construct BR$(\sigma^{\mathrm{U}}_{-i})$, because the solution π_1^2 was not contained in the component response policies. Instead, now with the value-based assumption, we have access to the value function for both response policies. This provides information about the quality of π_1^2 as it pertains to each of the opponent's policies, and can be used to exactly derive that π_1^2 is the best-response to σ^{U}_{-i}. As we'll explore in Section 5.3.1, this assertion holds true when state-actions, instead of policies, are the atomic units.

5.1 Normal-Form Game

To build up to a general solution, we first consider the simplified setting of a normal-form game, which notably has just a single state. The episode plays out by all players participating in one round of simultaneous action selection. All players then receive a single reward as their payoff and the episode concludes. The single-state setting is essentially a problem of bandit learning, where our opponent's strategy will set the reward of each arm for an episode.

Following from the bandit learning problem, intuitively, our expected reward against a mixture of opponents is proportional to the payoff against each opponent weighted by their respective likelihood. This insight motivates our method *Q-Mixing*, where we first train value-based best-responses to each opponent policy. Then we appropriately average the Q-values following the likelihood of playing against each policy within the mixture. To formalize this relationship, we introduce state- and action-values that are conditioned on fixed opponent strategies, called the Strategic Response Value (SRV) (Definition 3) and Strategic Response Q-Value respectively (SRQV) (Definition 4).

Definition 3 (Strategic Response Value). *An agent's* π_i strategic response value *is its expected return given an observation, when playing* π_i *against a specified opponent strategy:*

$$V_{\pi_i}(o_i^t \mid \sigma_{-i}^t) = \mathbb{E}_{\sigma_{-i}^t}\left[\sum_{\boldsymbol{a}} \boldsymbol{\pi}(a_i \mid o_i^t) \sum_{o_i', r_i} p(o_i', r_i \mid o_i^t, \boldsymbol{a}) \left[r_i + \gamma \cdot V_{\pi_i}(o_i' \mid \sigma_{-i}^{t+1})\right]\right].$$

Let the optimal SRV *be*

$$V_i^*(o_i^t \mid \sigma_{-i}^t) = \max_{\pi_i} V_{\pi_i}(o_i^t \mid \sigma_{-i}^t).$$

Definition 4 (Strategic Response Q-Value). *An agent's* π_i strategic response Q-value *is its expected return for an action given an observation, when playing* π_i *against a specified opponent strategy:*

$$Q_{\pi_i}(o_i^t, a_i^t \mid \sigma_{-i}^t) = \mathbb{E}_{\sigma_{-i}^t}\left[r_i^t\right] + \gamma \mathbb{E}_{o_i^{t+1}}\left[V_{\pi_i}(o_i^{t+1} \mid \sigma_{-i}^{t+1})\right],$$

where $r_i^t \equiv r_i(o_i^t, a_i^t, a_{-i}^t)$. *Let the* optimal SRQV *be*

$$Q_i^*(o_i^t, a_i^t \mid \sigma_{-i}^t) = \max_{\pi_i} Q_{\pi_i}(o_i^t, a_i^t \mid \sigma_{-i}^t).$$

Note, that the above definitions denote that the opponent's strategy is a mixed strategy σ_{-i}. When the opponent plays a pure strategy π_{-i} we can substitute the notation. For example, the SRQV against an opponent's first policy is $Q(\cdot \mid \pi_{-i}^0)$.

Q-Mixing captures the relationship between the SRQVs against a mixed-strategy opponent and the component SRQVs against each policy in the opponent's mixed strategy. In the single-state setting, weighting the SRQV against each opponent policy by the opponent's distribution supports a BR to that mixture. We define this relationship formally in Theorem 5, and refer to the single-state formulation as *Q-Mixing: Prior*.

Theorem 5 (Single-State Q-Mixing). *Let $Q_i^*(\cdot \mid \pi_{-i})$, $\pi_{-i} \in \Pi_{-i}$, denote the optimal strategic response Q-value against opponent policy π_{-i}. Then for any opponent mixture $\sigma_{-i} \in \Delta(\Pi_{-i})$, the optimal strategic response Q-value is given by*

$$Q_i^*(a_i \mid \sigma_{-i}) = \sum_{\pi_{-i} \in \Pi_{-i}} \sigma_{-i}(\pi_{-i}) \cdot Q_i^*(a_i \mid \pi_{-i}).$$

Proof. The definition of Q-value is as follows [Sutton and Barto, 2018]:

$$Q_i^*(a_i) = \sum_{r_i} p(r_i \mid a_i) \cdot r_i.$$

In a multiagent system, the dynamics model p suppresses the complexity introduced by the other agents. We can unpack the dynamics model to account for the other agents as follows:

$$p(r_i \mid a_i) = \sum_{\pi_{-i}} \sum_{a_{-i}} \pi_{-i}(a_{-i}) \cdot p(r_i \mid \boldsymbol{a}).$$

We can then unpack the strategic response value as follows:

$$Q_i^*(a_i \mid \pi_{-i}) = \sum_{a_{-i}} \pi_{-i}(a_{-i}) \sum_{r_i} p(r_i \mid \boldsymbol{a}) \cdot r_i.$$

Now we can rearrange the expanded Q-value to explicitly account for the opponent's strategy. The independence assumption enables the following re-writing by letting us treat the opponent's mixed strategy as a constant condition:

$$\begin{aligned} Q_i^*(a_i \mid \sigma_{-i}) &= \sum_{r_i} \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) \sum_{a_{-i}} \pi_{-i}(a_{-i}) \cdot p(r_i \mid \boldsymbol{a}) \cdot r_i \\ &= \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) \sum_{a_{-i}} \pi_{-i}(a_{-i}) \sum_{r_i} p(r_i \mid \boldsymbol{a}) \cdot r_i \\ &= \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) \cdot Q_i^*(a_i \mid \pi_{-i}). \end{aligned}$$

□

5.1.1 Didactic Example

		Player 2		
		π_2^{R}	π_2^{P}	π_2^{S}
	π_1^{R}	$0,\ 0$	$-1,\ 1$	$1,\ -1$
Player 1	π_1^{P}	$1,\ -1$	$0,\ 0$	$-1,\ 1$
	π_1^{S}	$-1,\ 1$	$1,\ -1$	$0,\ 0$

Figure 5.1: **Rock Paper Scissors.** Also known as “Roshambo.”

Consider the RPS game illustrated in Figure 5.1. Each episode consists of only a single state, then the agents simultaneously submit actions, and receive their rewards. From the perspective of Player 1, our opponent, Player 2, has the choice of three policies: rock π_2^{R}, paper π_2^{P}, and scissors π_2^{S}. Notice, that in this game policies are analogous to primitive actions, and that is not generally the case. For each of the opponent's policies we can determine the optimal SRQV by inspection:

$$Q_1^*(\cdot \mid \pi_2^{\text{R}}) = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \qquad Q_1^*(\cdot \mid \pi_2^{\text{P}}) = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \qquad Q_1^*(\cdot \mid \pi_2^{\text{S}}) = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}.$$

The SRQVs are found by first fixing the opponent's policy. In effect, the game is reduced to a 3×1 matrix game with known payoffs for each policy. This reduction removes any stochasticity introduced into payoff estimation that would result in sampling from a distribution of opponent policies. From the smaller matrix game, we need only consider our agent's payoffs (for Player 1 these are the first value in each cell of the game matrix).

Playing deterministically in RPS makes you an easily exploitable player. A stronger opponent may randomly choose between rock and paper yielding the mixed strategy $\sigma_{-i} = (0.5, 0.5, 0.0)$. Using single-state Q-Mixing we can compute the SRQV to said mixed strategy, assuming we know the mixture:

$$\begin{aligned}
Q_1^*(\cdot \mid \sigma_{-i}) &= \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) Q_1^*(\cdot \mid \pi_{-i}) \\
&= 0.5 \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} + 0.5 \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} + 0.0 \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} -0.5 \\ 0.5 \\ 0 \end{bmatrix}.
\end{aligned}$$

Therefore, we know that our optimal policy is π_1^{P}. When we play paper, we have the opportunity to win (unlike playing rock), and have no opportunity to lose (like scissors). The worst we can do is tie, making it our best-response.

5.2 Leveraging Information from the Past

Next, we consider enriching the previous setting to incorporate repeated interaction between the agents across an evolving observation distribution. The joint effect of the agents' actions influences this distribution and affords the opportunity to gather information about their opponent during an episode. Methods in this setting need to (1) leverage information from the past to update its' belief about their opponent, and (2) grapple with the uncertainty about the future. Accordingly, extending Q-Mixing into this setting requires quantification of the agent's current belief about their opponent and their future uncertainty.

We will begin by focusing on the first condition: the method must leverage information from the past to update its' belief about their opponent. When compared to the single-state setting, each agent now has access to a history of their observations $\mathcal{O}_i^{0:t}$. Additionally, we will not presently take into consideration that the agent may gain future evidence about the identity of their opponent's policy (see Section 5.3). In essence, the current setting reflects that of the penultimate state of a game. Where each agent has all of the previous play to consider; however, they know that they are deciding their final action for this episode.

During an episode the actual observations experienced generally depend on the identity of the opponent's policy, which is drawn from their mixed strategy. Let

$$\psi_i : \mathcal{O}_i^{0:t} \to \Delta(\Pi_{-i}) \tag{5.1}$$

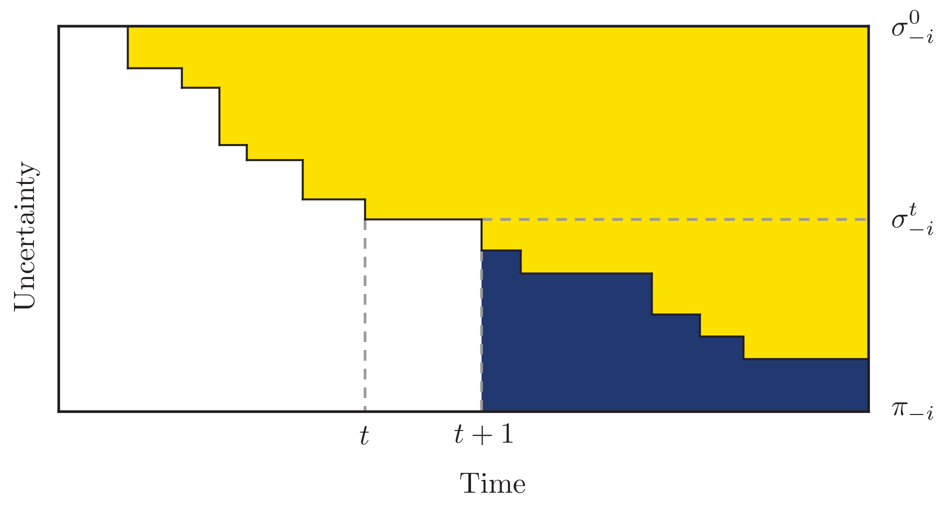


Figure 5.2: **Conceptual opponent uncertainty over time.** The yellow area represents a uncertainty reduction, for some measure, as a result of updating belief about the distribution of the opponent. The blue area represents approximation error incurred by Q-Mixing.

represent the agent's current belief about the opponent's policy using the observations during play as evidence. From this prediction, we propose an approximate version of Q-Mixing that accounts for past information. The approximation works by first predicting the relative likelihood of each opponent policy given the current observation. Then it weights the Q-value-based BRs against each opponent by their relative likelihood.

Figure 5.2 provides a conceptual illustration of the benefits and limitations of this new prediction-based Q-Mixing. At any given timestep t during the episode, the information available to an agent about the opponents may be insufficient to perfectly identify their policy. Nevertheless, the agent maintains a belief σ^t_{-i} of the identity of their opponent's policy. The yellow area above a timestep represents the uncertainty reduction from an updated prediction of the opponent σ^t_{-i} compared to the baseline prediction of the prior σ^0_{-i}. Crucially, this definition of Q-Mixing does not consider updating the opponent distribution from new information in the future (blue area in Figure 5.2).

Let the previously defined ψ be the *opponent policy classifier* (OPC), which predicts the identity of the opponent policy. We then augment Q-Mixing to weight the importance of each BR as follows:[1]

$$Q_{\pi_i}(o_i, a_i \mid \sigma_{-i}) = \sum_{\pi_{-i}} \psi_i(\pi_{-i} \mid o_i, \sigma_{-i}) Q_{\pi_i}(o_i, a_i \mid \pi_{-i}). \tag{5.2}$$

We refer to this quantity as *Q-Mixing*, or *Q-Mixing: X*, where X describes ψ. The version of

[1] I suppress the notation that observations include histories of observations. This is for both ease of reading and because the requisite amount of history required will vary per game.

Q-Mixing introduced in the single-state setting will be therefore referred to as *Q-Mixing: Prior*, where the prior belief σ^0_{-i} is the known mixed strategy of the opponent. By continually updating the opponent distribution during play, the adjusted Q-Mixing result better responds to the actual opponent.

An ancillary benefit of the opponent classifier is that poorly estimated Q-values tend to have their impact minimized. For example, if an observation occurs only against the second opponent policy, then the Q-value against the first opponent policy would not necessarily be trained well, and thus could distort the policy from Q-Mixing. These poorly trained cases correspond to unlikely opponents and get reduced weighting in the version of Q-Mixing augmented by the classifier.

5.2.1 Running With Scissors

We first evaluate Q-Mixing on the RWS grid-world game [Vezhnevets et al., 2020, Leibo et al., 2021], detailed in Section 2.6.1. With this environment we pose the following questions:

1. Can Q-Mixing transfer Q-values from pure strategy responses to generate *a* mixed strategy response?
2. Is Q-Mixing capable of transferring Q-values across *all* mixed strategies?
3. Does incorporating an OPC that updates the opponent distribution in Q-Mixing enhance its performance?

In our experiments, we assume the perspective of Player 1 and learn LSTM-based [Hochreiter and Schmidhuber, 1997] response policies using Double DQN [van Hasselt et al., 2016]. The state space is a one-hot encoding of the 10 possible occupants of each cell. Resulting in a ravelled vector with length 253, including observation of the player's inventory. The agent has the option of selecting one of 9 actions: move in the four cardinal directions, rotate left or right, challenge (fire/beam) their opponent, or to take no action. The LSTM has a memory size of 128, and the output is projected through a series of fully-connected layers with sizes [128, 64, 64, 9].

We learn three different policies for Player 2 that are then fixed for evaluation. Each of these policies is specialized to prefer collecting one of the three items. To train such a policy, an auxiliary reward of 1 is added each time the agent collects their preferred item.

5.2.2 Transfer Onto A Mixed Strategy

With our game and opponents established, we can address our first research question: can Q-Mixing effectively transfer Q-values from pure strategy responses to generate a mixed strategy response? To assess this, we examine the method's effectiveness in generating a response policy

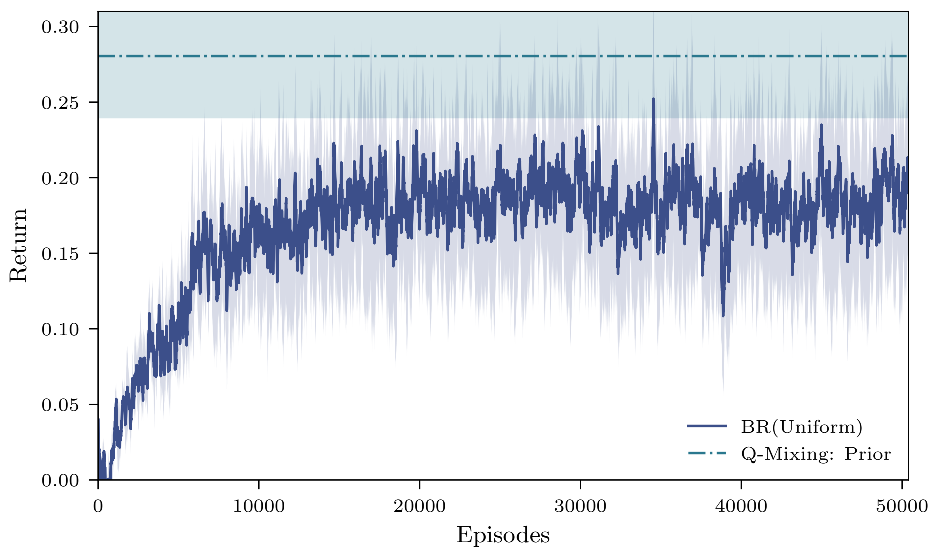


Figure 5.3: **BR(Uniform)'s learning curve compared to Q-Mixing.** BR(Uniform)'s performance is reported in terms of a sliding window of return over the last 100 episodes. Q-Mixing transfers BRs the the opponent's policies that were trained using equal fractions of the training budget available to BR(Uniform). Q-Mixing is then evaluated by simulating its return against each opponent policy for 100 episodes. Both methods' performances are reported including a 95% bootstrap confidence interval.

to a uniform mixed strategy composed of three opponent policies. As a baseline, we use a best-response policy trained directly against the mixed strategy, denoted as BR(Uniform). Constructing a Q-Mixing policy first requires training best-responses directly against the individual opponent policies. Simultaneously, best-responses are created directly against the individual opponent policies. These pure strategy best-responses employ the same neural network architecture but divide the simulation budget allotted to BR(Uniform) evenly. By allocating the simulation budget in this manner, we account for outcomes potentially influenced by access to larger amounts of simulation data. The responses thus obtained are then utilized to construct the Q-Mixing: Prior policy.

We plot BR(Uniform)'s training curve in Figure 5.3. On this plot, we also include the simulated performance of Q-Mixing: Prior. It is important to note here that Q-Mixing: Prior requires having previously trained best-responses to each of the opponents individual policies. Since we are only investigating here the quality of this transfer operation, we do not account for this prior simulation time. This disclaimer in mind, we find that Q-Mixing: Prior is able to successfully transfer Q-values to generate a successful best-response policy. In fact, the Q-Mixing: Prior policy outperforms to BR(Uniform) without requiring any addition training against the objective. This performance gap can be potentially explained by

(a) benefits from specialization, and

(b) limitations of RL as an approximate best-response oracle.

Q-Mixing allows best-responses to be trained directly against each opponent policy, disentangling the belief in the opponent's policy from the value of each action. As a result, the Q-values need not be risk-averse and choose sub-optimal responses, which is crucial for BR(Uniform) as it must concern itself with the other opponent policies. As for the limitations of RL as an approximate best-response, we accept this limitation because any approximate method will have its drawbacks. Moreover, both methods were treated with equal hyperparameter tuning and training budget (measured in training timesteps). A practitioner should prefer Q-Mixing, because this result suggests it is much easier to produce quick and strong results (at least for this game). In summary, Q-Mixing shows efficacy in transferring strategic knowledge from opponent policies onto an opponent mixed strategy, confirming our first research question.

5.2.3 Opponent Strategy Space Coverage

The previous result suggests that Q-Mixing may be particularly advantageous when we need to repeatedly generate responses to differing opponent strategies. We investigate this possibility in our second research question. Can Q-Mixing transfer Q-values across *all* of the opponent's mixed strategies?

In order to investigate this question we evaluate the same two methods against a representative coverage of the entire strategy space of the opponent. The strategy sets we consider are all mixtures truncated to the tenths place (e.g., $[0.3, 0.4, 0.3]$). Q-Mixing methods depend on the changing opponent mixed strategy, unlike BR(Uniform), which is unchanged across opponent strategies. Therefore, when evaluating BR(Uniform) we simulate its performance against each respective opponent policy for 300 episodes[2]. Then the expected performance against each mixed strategy can be calculated by appropriately averaging the mean returns against the respective policies. As for Q-Mixing, we must simulate the performance against each opponent mixture independently. Q-Mixing must be updated to condition on each opponent mixture, and then be simulated against each opponent policy for 300 episodes.

We evaluate both methods based on their return and their *normalized return*. The normalized return, normalizes an estimated return against an opponent policy by the return received by its respective BR. The normalized returns are then averaged according to the opponents mixture, as follows:

$$\|u(\pi_i, \sigma_{-i})\| = \frac{\sum_{\pi_{-i} \in \sigma_{-i}} \sigma_{-i}(\pi_{-i}) \cdot u(\pi_i, \pi_{-i})}{\sum_{\pi_{-i} \in \sigma_{-i}} \sigma_{-i}(\pi_{-i}) \cdot u(\text{BR}(\pi_i), \pi_{-i})} \tag{5.3}$$

Looking at both performance metrics allows us to more fairly compare the distribution of returns

[2]The number of episodes was chosen because the mean return empirically converged by 300 episodes.

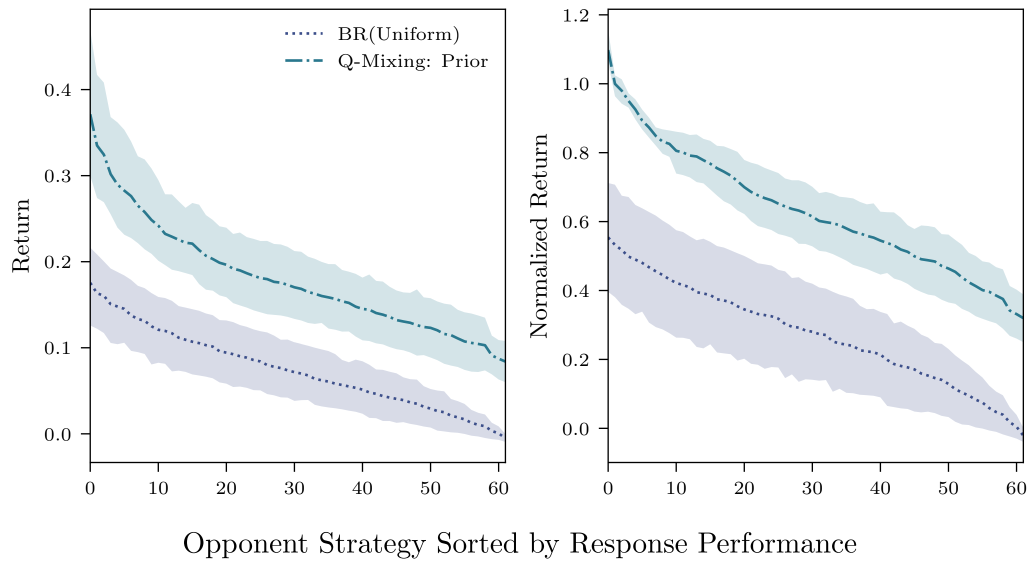


Figure 5.4: **Coverage of Q-Mixing: Prior on RWS.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy.

across different opponent policies. For example, consider an opponent with two policies, the later being vulnerable to dramatic exploitation. If we estimated our returns against these policies as 1 and 1000 respectively, then any evaluations against a mixture of these two policies would not fairly account for the performance to the first opponent policy.

Figure 5.4 shows Q-Mixing's performance across the opponent mixed strategy space. As we can see in both plots, Q-Mixing strictly dominates the performance of BR(Uniform). A Q-Mixing method that perfectly transfers the component BR knowledge would have a curve that is a horizontal line at 1. This upper bound is unrealistic in practice, because it effectively requires perfectly identifying the opponent policy prior to play. Nevertheless, the difference between Q-Mixing: Prior and 1 represents the potential room for improvement. An astute reader then may wonder how is it possible for Q-Mixing: Prior to achieve a performance greater than 1? This results is from the serendipitous circumstance where the Q-values from the other responses offer advantageous information when weighted together.

However, Q-Mixing requires access to the opponent distribution, an assumption that can potential limit its applicability. We will see below, alternative instantiations of Q-Mixing that may relax this assumption.

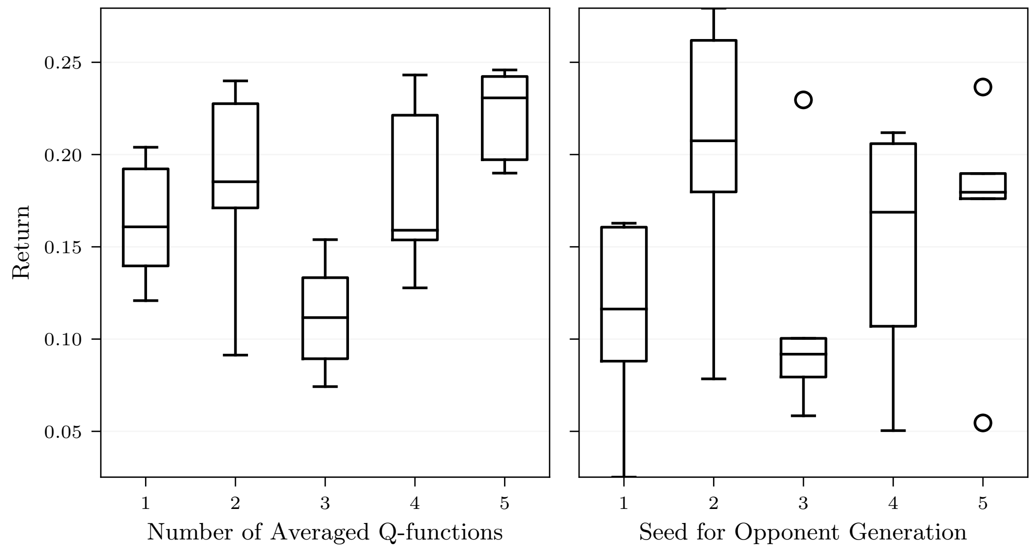


Figure 5.5: **Effects of averaging Q-values from DQN.** (Left) performance of averaging the Q-values from an increasing number of DQNs against the uniform mixed-strategy opponent. (Right) the respective performance of the individual DQNs.

5.2.4 Q-value Regularization

A possible explanation for the performance improvements observed in Q-Mixing is that it benefits from value ensembling. Value ensembling has been shown to reduce noise in BR learning, leading to a more stable training target [Anschel et al., 2017]. In the context of Q-Mixing, this would suggest that the advantage is derived from the aggregation of value predictors, rather than the specialization of per-opponent policy responses or their weighting. To determine if regularization is the cause of our previous findings, we examine the benefits of regularization in this domain using a set of independently trained Q-functions. If averaging these Q-functions corresponds to performance improvements, then regularization could be a plausible explanation for the performance improvements seen in Q-Mixing.

In Figure 5.5, we show the performance of uniformly averaging the Q-values from an increasing number of the independently trained DQNs. There appears to be no consistent trend in performance improvement or degradation as additional DQNs are introduced. Going from one to two DQNs results in an improvement, but adding a third DQN eliminates the previous gains and further reduces performance. In the same figure, we show the performance of each DQN in isolation. The individual performance allows us to better understand the regularization results: changes in regularization performance coincided with the relative performance of each newly added Q-function.

Next, we explore whether any trend emerges when regularizing the outputs of two DQNs together. This experiment is inspired by the hypothesis that the order in which DQNs are added

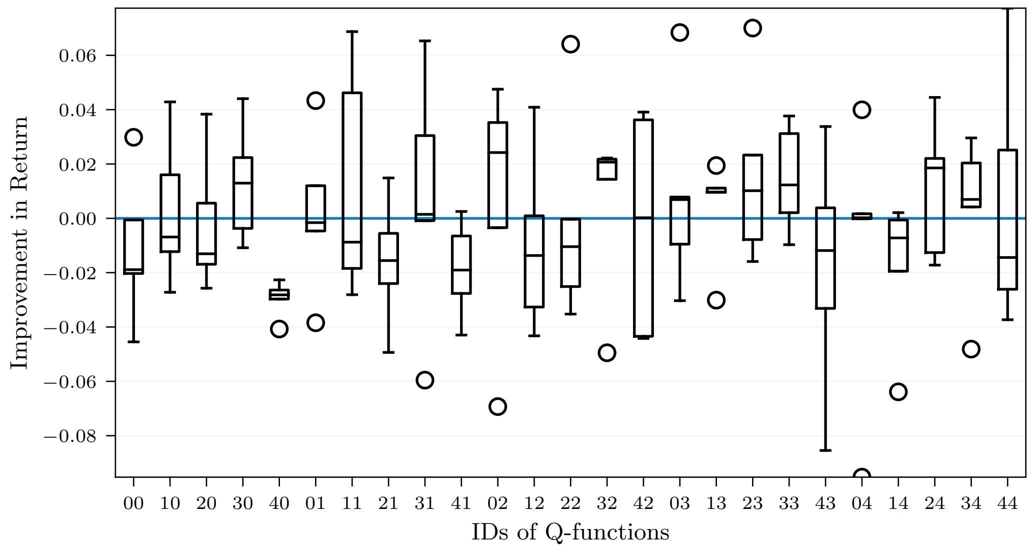


Figure 5.6: **Performance improvement of averaging an additional DQN.** The x-axis denotes the IDs of the baseline DQN (first index), and the additional DQN (second index) added to regularize its' Q-values.

during evaluation may have confounded our previous results when regularizing a group of DQNs. In Figure 5.6, we plot the improvement in return when an additional DQN is averaged with a baseline DQN. 12 of the 25 combinations saw an improvement in performance from the addition of another Q-value estimate. In conjunction with the previous results, we find no compelling evidence to suggest that Q-value regularization is the primary factor contributing to the benefits of Q-Mixing.

5.2.5 Opponent Classification

Our third research question is: can the use of an OPC that updates the opponent distribution in Q-Mixing improve its performance? During play against an opponent sampled from the mixed strategy, the player is able to gather evidence about which opponent they are playing. We hypothesize that leveraging this evidence to weight the importance of the respective BR's Q-values higher will improve Q-Mixing's performance.

To verify this hypothesis, we train an OPC using the replay buffers associated with each BR policy. These are the same buffers that were used to train the BRs, and cost no additional compute to collect. This data is used to train an OPC that outputs a distribution over opponent pure strategies for each observation. The OPC is implemented with a deep neural network with the same architecture as the policies, save the last layer that has size that equals the number of opponent policies (in this case, 3). The classifier is trained to predict from an observation, sampled across the replay buffers, the respective pure strategy index it occurred against with a cross-entropy loss.

We evaluate Q-Mixing: OPC by testing the performance on a representative coverage of the mixed-strategy opponents illustrated in Figure 5.7. In order to compute coverage curves we follow two methodologies based off whether the response policy can leverage knowledge of the opponent's strategy. For methods where the response policy cannot use the opponent's strategy (e.g., BR(0) cannot benefit from being told they are playing opponent policy 1), the performance against each opponent policy does not change across strategies. Therefore, we first simulate the performance against each opponent policy, then we can compute the performance against each opponent strategy by appropriately averaging the respective performance against each opponent policy by the likelihood of playing said opponent. On the other hand, methods such as Q-Mixing can condition on the opponent's strategy generating a unique response policy per for each opponent strategy. To evaluate these methods, we must simulate the response policy's performance against each opponent strategy independently. Keeping with a similar sampling procedure as the first methodology, we first simulate the per opponent policy performance and then average these performances by the opponent's strategy. However, this must be uniquely computed for each opponent strategy.

We found that the Q-Mixing: OPC policy performed stronger against the full opponent strategy coverage than Q-Mixing: Prior. This result supports our hypothesis that an OPC can identify the opponent's pure strategy and enable Q-Mixing to chose the correct BR policy. However, Q-Mixing: OPC method has a larger variance in return. This variance is not found in the normalized return, suggesting that the larger variance is a result of the range of exploitability of the respective opponent policies. Previously, this trend in variance was thought to be a result of opponent misclassification.

One limitation of this particular instance of the OPC is the need for retraining the classifier whenever there are changes in the set of opponent policies. However, the cost of training the OPC is relatively low compared to training a policy. This is primarily because the training procedure for the OPC, which is a supervised learning process, does not require interaction with a simulator to generate training data. This characteristic makes the OPC method viable, but it also underscores a potential direction for future research: the development of an opponent likelihood model capable of handling a dynamic set of opponent policies.

5.2.6 Policy Distillation

In Q-Mixing we need to compute Q-values for each of the opponent's pure strategies. This can be a limiting factor in parametric policies, like deep neural networks, where our policy's complexity grows linearly in the size of the support of the opponent's mixture. This can become unwieldy in both memory and computation. To remedy these issues, we propose using policy distillation to

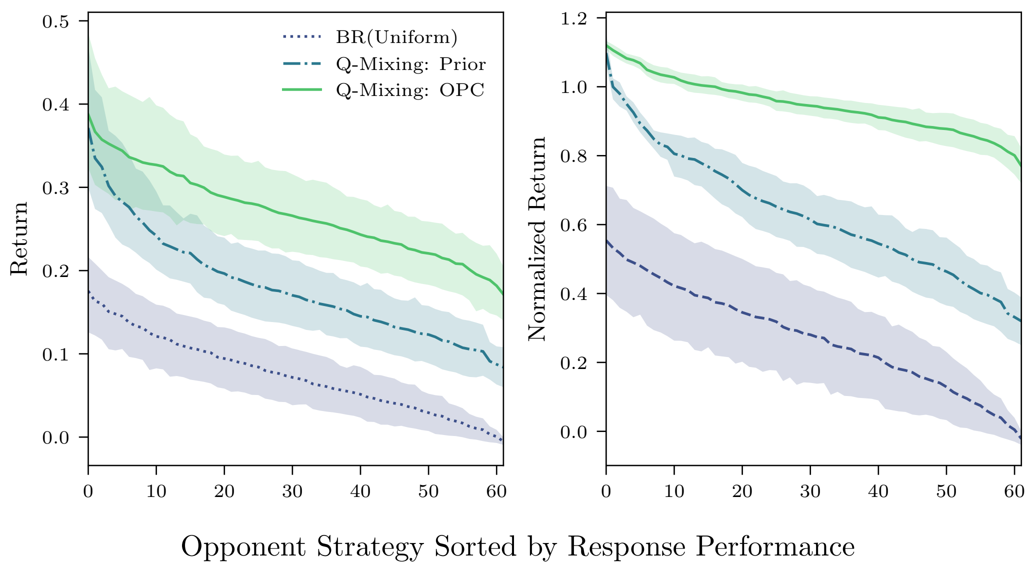


Figure 5.7: **Coverage of Q-Mixing: OPC on RWS.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy.

compress a Q-Mixing policy into a smaller parametric space [Hinton et al., 2014, Rusu et al., 2015, Buciluǎ et al., 2006].

In the policy distillation framework, a larger neural network referred to as the *teacher* is used as a training target for a smaller neural network called the *student*. In our experiment, the Q-Mixing policy is the teacher to a student neural network that is the size of a single best-response policy. The student is trained in a supervised learning framework, where the dataset is the concatenated replay buffers from training pure-strategy best-responses. This is the same dataset that was used in opponent classifying, which was notably generated without running any additional simulations. A batch of data is sampled from the replay-buffer and the student predicts Q^S the teacher's Q^T Q-values for each action. The student is then trained to minimize the KL-divergence between the predicted Q-values and the teacher's true Q-values. There is a small wrinkle, the policies produce Q-values, and KL-divergence is a metric over probability distributions. To make this loss function compatible, the Q-values are transformed into a probability distribution by softmax with temperature τ. The temperature parameter allows us to control the softness of the maximum operator. A high temperature produces actions that have a near-uniform probability, and as the temperature is lowered the distribution concentrates weight on the highest Q-Values. The benefit of a higher temperature is that more information can be passed from the teacher to the student about each state.

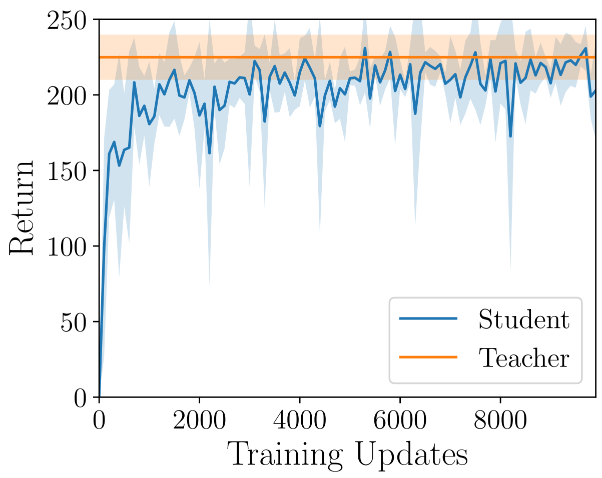


Figure 5.8: **Policy distillation simulation performance over training on the Gathering game.** The teacher Q-Mixing-Prior is used as a learning target for a smaller student network. Shaded region represents a 95% confidence interval over five random seeds ($df = 4,\ t = 2.776$).

The learning curve of the student is reported in Figure 5.8. We found that the student policy was able to recover the performance of Q-Mixing-Prior, albeit with slightly higher variance. This study did not include any effort to optimize the student's performance, thus further improvements with the same methodology may be possible. This result confirms our hypothesis that policy distillation is able to effectively compress a policy derived by Q-Mixing.

5.3 Accounting for Future Uncertainty

Q-Mixing as we have seen it so far can handle the timeless setting (single-state), and consider information from the past and present. So far, the belief in the opponent's policy is assumed constant into the future. However, the future offers additional opportunities to gather evidence that may influence the belief in the opponent's policy. And in these future states, the belief at that point should be updated to reflect the cumulative evidence gathered about the opponent's policy.

5.3.1 Opponent-Policy Identification Game

To illustrate this important detail we introduce the *opponent-policy identification game*. This game, illustrated in Figure 5.9, is a form of coordination game where an agent has the option to pay a cost to observe the opponent's policy prior to coordination. Success in this game requires that the agent can appropriately trade-off the cost of information gathering with the benefit of a more informed future belief in the opponent's policy.

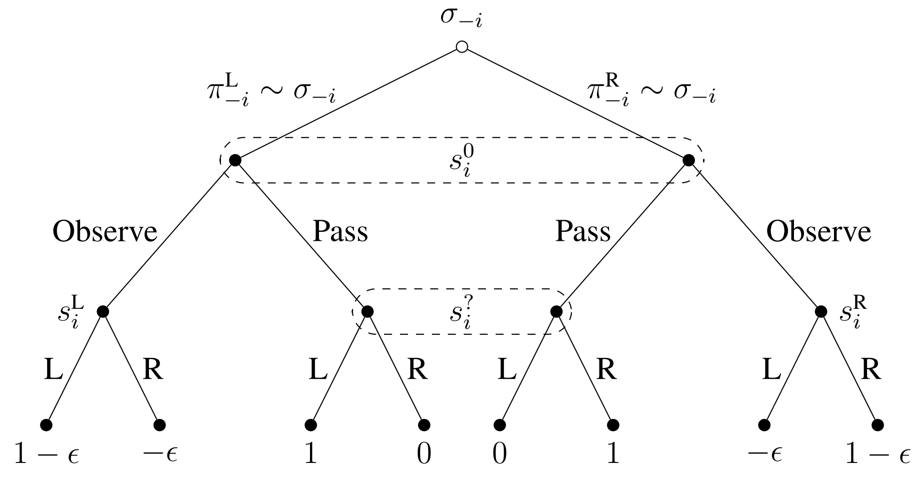


Figure 5.9: **Opponent-policy identification game.** Leaf node values reflect the full return for player i.

At the root, the opponent draws a policy from their mixed strategy $\sigma_{-i} \in \Delta(\{\pi^{\text{L}}_{-i}, \pi^{\text{R}}_{-i}\})$. Player i begins in initial state s_i^0, where they have no information about the opponent's policy. From this state, player i has the option to observe the opponent's identity—that is, whether it chose the L or R policy—for a cost of ϵ. If they exercise this ability, they transition to a state of knowing the opponent's identity $\{s_i^{\text{L}}, s_i^{\text{R}}\}$ (where the superscript corresponds to the opponent's policy). If they pass on this opportunity, then they remain in a state of ignorance $s_i^?$.

After player i has observed or passed, the player chooses L or R. If i's choice matches the opponent's policy label, the reward to both players is 1; otherwise they receive no reward. The total return, or payoff, for player i in this game is then the cost of observation, if exercised, plus the reward conditional on successful coordination.

The BR to an opponent in this game depends on both the opponent's strategy σ_{-i} and the cost of observation ϵ. For instance, consider the uniform mixed strategy opponent with a small observation cost:

$$\sigma^{\text{U}}_{-i} \leftarrow \text{Uniform}(\{\pi^{\text{L}}_{-i}, \pi^{\text{R}}_{-i}\}) \qquad \epsilon \leftarrow 0.03.$$

The best-response in this setting, BR(σ^{U}_{-i}), is to observe the opponent's policy and play the appropriate response. Consequently, BR(σ^{U}_{-i}) receives a return of 0.97. Had this response policy chosen to not observe the opponent's identity, then they can do no better than chance during coordination and would receive a return of 0.5.

Now, let us consider our transfer learning problem within the aforementioned setting of the opponent-policy identification game. The problem asks us to construct BR(σ^{U}_{-i}) from

$$\{\text{BR}(\pi^{\text{L}}_{-i}), \text{BR}(\pi^{\text{R}}_{-i})\},$$

given σ_{-i}. The pure strategy best-responses in effect already know the identity of the opponent. This means that they never decide to observe the identity, as doing so incurs an unnecessary cost without providing any additional information. Hence, we can summarize the pure strategy responses as follows (the BRs are rewritten as conditioned policies for ease of notation):

$$\begin{aligned} \pi_i(s_i^0 \mid \pi_{-i}^{\mathrm{L}}) &= \text{Pass} & \qquad \pi_i(s_i^0 \mid \pi_{-i}^{\mathrm{R}}) &= \text{Pass} \\ \pi_i(s_i^? \mid \pi_{-i}^{\mathrm{L}}) &= \mathrm{L} & \qquad \pi_i(s_i^? \mid \pi_{-i}^{\mathrm{R}}) &= \mathrm{R}. \end{aligned}$$

From these component policies we cannot construct $\mathrm{BR}(\sigma_{-i}^{\mathrm{U}})$, because it contains a new strategic behavior not present in the provided responses: the need to gather additional information about the opponent. As we saw in the problem setup, $\mathrm{BR}(\sigma_{-i}^{\mathrm{U}})$ receives a higher return if it chooses to take the observe action and appropriately respond. The information-gathering behavior fundamental to the correctness of a mixed strategy response is, in this example, not present in responses to the opponent pure strategies.

In the general opponent-policy identification game, we can successfully transfer responses when acquiring additional information on the identity of the opponent is not worthwhile. Information-gathering is not worthwhile when

$$1 - \epsilon < \max_{\pi_{-i}} \sigma_{-i}(\pi_{-i}),$$

where $\sigma_{-i}(\pi_{-i})$ is the probability of the opponent playing π_{-i}. If this inequality holds, then the cost for observation outweighs its benefit. Conversely, if the inequality is reversed, then $\mathrm{BR}(\sigma_{-i}^{\mathrm{U}})$ stands to benefit from taking actions to acquire knowledge about the opponent's identity. This information-gathering behavior will not be evident in the component response policies, because it is not optimal for them to incur the information-gathering cost. As a result, $\mathrm{BR}(\sigma_{-i}^{\mathrm{U}})$ cannot be constructed without injecting additional strategic knowledge related to opponent policy identification.

5.3.2 Q-Mixing with Value Iteration

To account for future uncertainty, Q-Mixing must be able to update its successor observation values given future evidence of the opponent's policy. This can be done by expanding the Q-value into its components: expected reward under the current belief in the opponent's policy, and our expected next observation value. By updating the second term to recursively reference a new opponent belief we can account for changing beliefs in the future. The extended formulation, Q-Mixing:

Value Iteration (QMVI), is given by:

$$Q_i^*(o_i^t, a_i^t \mid \sigma_{-i}) = \sum_{\pi_{-i} \in \Pi_{-i}} \psi_i(\pi_{-i} \mid o_i^t, \sigma_{-i}) \cdot \Big[r_i(o_i^t, a_i^t \mid \pi_{-i}) + \gamma \mathbb{E}_{o_i^{t+1}} \big[V^*(o_i^{t+1} \mid \sigma_{-i}) \big] \Big] . \quad (5.4)$$

If we assume that we have access to both a dynamics model and the observation distribution dependent on the opponent, then we can directly solve for this quantity through Value Iteration (Algorithm 2). These requirements are quite strong, essentially requiring perfect knowledge of the system with regards to all opponent policies. The additional step of Value Iteration also carries a computational burden, as it requires iterating over the full state and action spaces. Though these costs may render QMVI infeasible in practice, we provide Algorithm 2 below as a way to ensure correctness in Q-values.

Algorithm 2: Value Iteration: Q-Mixing

```
Input: $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \epsilon, \gamma$
$V_0(s \mid \sigma_{-i}) \leftarrow \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) Q(s, a \mid \pi_{-i})$
do
    $Q_t(s, a \mid \sigma_{-i}) \leftarrow \sum_{\pi_{-i}} \psi(\pi_{-i} \mid s, \sigma_{-i}) \sum_{s',\, r} \mathcal{T}(s', r \mid s, a, \pi_{-i})[r + \gamma V_{t-1}(s' \mid \sigma_{-i})]$
    $V_t(s \mid \sigma_{-i}) \leftarrow \max_a Q_t(s, a \mid \sigma_{-i})$
    $\pi_t(s \mid \sigma_{-i}) \leftarrow \arg\max_a Q_t(s, a \mid \sigma_{-i})$
while $\exists_{s \in \mathcal{S}} |V_t(s) - V_{t-1}(s)| > \epsilon$
Output: $V_t,\ Q_t,\ \pi_t$
```

QMVI is reducible into the traditional value iteration algorithm. Consider we construct a new aggregate MDP by combining both the original MDP and the opponent's dynamics (represented by the combination of their strategy and policies). The aggregated MDP is both stationary and may be stochastic, because the opponent is not learning and may randomize their play. Convergence properties of value iteration are inherited by QMVI by considering the application of value iteration on said aggregate MDP.

Throughout the remainder of this dissertation, I will employ Q-Mixing methods that do not account for future uncertainty. While the experiments above have demonstrated that these methods are empirically performant, they are limited in the correctness of the values they compute. Understanding the discrepancy between these methods and those that accurately anticipate the future is crucial for assessing the reliability of Q-Mixing. Future work should also consider stopgap methods, that trade-off the computational demands required by exact methods with approximate methods that are more efficient.

5.4 Related Work

Traditional RL algorithms include no mechanism to explicitly prepare for variability in opponent mixture. Instead, current solutions either learn a new behavior or update a previously learned behavior. in lieu of that, researchers designed methods for learning a single behavior successfully across a set of strategies [Wang and Sandholm, 2003], or quickly adapting in response to new strategies [Jaderberg et al., 2019].

Alternative multiagent RL methods build a predictive model of their opponent and use this to inform their decision making processes. He et al. [2016] introduce DRON, which uses a learned latent action prediction of the opponent as conditioning information to the policy (in a similar nature to the opponent-actions in the joint-action value area). They also present a variant of DRON that uses a Mixture-of-Experts [Jacobs et al., 1991] operation to marginalize over the possible opponent behaviors. More formally, they compute the expected Q-value by marginalizing the opponent's action space:

$$\sum_{a_{-i}} \pi_{-i}(a_{-i} \mid s_i) Q_i(s_i, a_i, a_{-i}). \tag{5.5}$$

Q-Mixing employs a similar style of operation; however, it marginalizes over the policy-space of the opponent instead of the action-space. Moreover, Q-Mixing depends on independent BR Q-values against each opponent policy, whereas DRON learns a single Q-network. Bard et al. [2013] propose implicitly modeling opponents through the payoffs received from playing against a portfolio of the agent's policies.

Q-Mixing also bears resemblance to Bayesian Policy Reuse (BPR) [Rosman et al., 2016]. Both methods leverage a library of precomputed policies; however, BPR performs single-agent task identification to select which policy to play during each episode, whereas, Q-Mixing maintains an online belief and uses this to compute Q-values. The family of PEPPER algorithms have investigated applying BPR to games [Crandall, 2012]. This line of work primarily focuses on identification of opponent-policy switching during a single play (episode) [Hernandez-Leal and Kaisers, 2017a,b]; whilst, this study assumes that the opponent policy is constant for the full duration of each episode.

The multi-task community has also separately explored approaches that have a similar style of machinery. Progressively growing neural networks is a similar line of work [Rusu et al., 2016], focused on a stream of new tasks. In our multiagent scenario, the stream of new tasks is learning responses to each opponent policy independently. They then ensemble these networks together for inference; however, this operation shares no commonalities with Q-Mixing. Schwarz et al. [2018] found that this ensembled network growth could be handled with policy distillation.

CHAPTER 6

Opponent Policy Likelihood

I have no cramps and I feel strong. It is he that has the hook in his mouth. But what a fish to pull like that. He must have his mouth shut tight on the wire. I wish I could see him. I wish I could see him only once to know what I have against me.

— Ernest Hemingway, *The Old Man and the Sea*

As we have seen from both policy- and value-based strategic knowledge transfer, a crucial component to their success is the ability to identify their opponent's policy. It comes at no surprise that response knowledge against a particular opponent policy may prove fruitless if you cannot reasonably expect that you are interacting with that opponent policy. With this in mind, we turn our attention towards the problem of modeling the agent's belief of their opponent's current policy.

Correctly identifying your opponent's policy facilitates playing the appropriate best-response. The identification problem is typically not straightforward due to the limited information that can be observed about your opponent's strategy. Reasonably, it will not be the case that the opponent will tell their competitor their policy. Instead, one must collect information about their opponent and use this to inform a belief of the opponent's policy.

Information about the opponent's strategy may be collected prior to gameplay and during interaction with a particular policy. The former information informs one's *prior* on the opponent's policy. Whereas, the later pertains to *evidence* gained to inform the *likelihood* of the current opponent policy. The likelihood may be calculated with only the evidence at the current timestep, or may use the full *history* of evidence gained from the start of the episode until the present. A prior and history-based likelihood model together constitute a fully informed belief model. This raises a key question: what components to the belief model are critical to its success?

Before we answer this question, we must first understand what makes a good likelihood and prior. In the preceding section we demonstrated the efficacy of two methods for maintaining a belief of the opponent's current policy:

1. the prior defined by the opponent's mixed strategy, and
2. a neural network likelihood model (trained through the auxiliary task of classification).

Both of these methods failed to fully take advantage of all of the information available; in fact, each method lacks what the other provides. The prior method fails to account for evidence during play with a particular opponent policy, and the classifier fails to be amenable to changes in the prior belief of the opponent policy. Moreover, both previous options do not consider any historical evidence in their belief calculation.

6.1 Opponent Likelihood Model

We begin by investigating the design of an opponent likelihood model. In the previous section, we trained a neural network classifier to predict which opponent we are playing against based on the game history. Assuming a well calibrated model, this will accurately predict the posterior probability of each opponent, but only if the distribution over opponent policies in the data used to train the model was the same as the opponent's strategy used in Q-Mixing. If the distribution of the data was to differ from the data experienced by the Q-Mixing policy, we would need to correct for this change.

Let $\bar{\sigma}$ be the opponent mixture used to train the OPC and $h \in \mathcal{H}$ be any realizable history, then we can define and unpack OPC ψ by Bayes Theorem as follows (up to model approximation errors):

$$\begin{aligned}\psi(\pi \mid h) &= \frac{p(h \mid \pi) \cdot \bar{\sigma}(\pi)}{p(h \mid \bar{\sigma})} \\ &\propto p(h \mid \pi) \cdot \bar{\sigma}(\pi). \qquad (6.1)\end{aligned}$$

This results in two terms directly corresponding to the likelihood and prior of our opponent's identity. Previously, when investigating the opponent identification, a neural network was trained to implicitly model this distribution through classification. A prior on the opponent's policy is implicitly trained into the OPC through the class balance $\bar{\sigma}$ in the dataset. In other words, if the data used to trained the OPC contained mostly experience against opponent two, then this may bias the OPC to favor predicting that opponent. In the previous experiments the data was balanced across classes; following this, the OPC adopted a uniform prior.

When using Q-Mixing in a training procedure (such as in Chapter 7) to respond to a mixed-strategy opponent, our agent has access to the opponent's mixture. The OPC fails to take advantage of this new prior knowledge; the distribution used to train the likelihood model now fails to match the true opponent distribution. However, because the class distribution in training $\bar{\sigma}$ is known, rather than using the OPC directly, we can treat it as an Opponent Likelihood Model (OLM), up to a multiplicative constant.

Instead of assuming the same prior $\bar{\sigma}$ across the entire opponent strategy-space we would like to update ψ, our likelihood model, when given new information about the prior distribution of the opponent policies. Consider a different mixed strategy σ, we would like to update ψ so that it instead approximates the distribution:

$$\begin{aligned} p(\pi \mid h, \sigma) &= \frac{p(h \mid \pi) \cdot \sigma(\pi)}{p(h \mid \sigma)} \\ &\propto p(h \mid \pi) \cdot \sigma(\pi). \end{aligned} \tag{6.2}$$

By rearranging Equation 6.1 we get the likelihood in terms of our trained model:

$$p(h \mid \pi) \propto \frac{\psi(\pi \mid h)}{\bar{\sigma}(\pi)}. \tag{6.3}$$

Then we may substitute Equation 6.3 into Equation 6.2, facilitating a correction for the prior:

$$\begin{aligned} p(\pi \mid h, \sigma) &\propto \frac{\psi(\pi \mid h)}{\bar{\sigma}(\pi)} \sigma(\pi) \\ &= \psi(\pi \mid h) \frac{\sigma(\pi)}{\bar{\sigma}(\pi)} \end{aligned} \tag{6.4}$$

This new formulation addresses the concerns we raised earlier. First, we account for prior knowledge of the opponent by directly using our prior σ. Second, we can learn a likelihood model of the evidence h through our previous method for constructing an OPC. Finally, we correct for the prior used during the training of the evidence likelihood.

By substituting the method used to compute likelihoods in Q-Mixing, we conduct an ablation study to examine the impact of both the new prior knowledge and the evidence obtained during gameplay. The respective impacts are assessed by establishing four version of Q-Mixing, as listed in Table 6.1, each varying in the information sources that inform their likelihood calculations. The version of Q-Mixing developed in this section, incorporating both sources of information, is hereafter referred to as *Q-Mixing: OLM*:

$$Q_{\pi_i}(h_i, a_i \mid \sigma_{-i}) \propto \sum_{\pi_{-i}} \psi_i(\pi_{-i} \mid h_i) \frac{\sigma_{-i}(\pi_{-i})}{\bar{\sigma}_{-i}(\pi_{-i})} Q_{\pi_i}(h_i, a_i \mid \pi_{-i}). \tag{6.5}$$

This version of Q-Mixing is constructed derived from substituting the classifier-based likelihood in Equation 5.2 with the corrected likelihood in Equation 6.4. The likelihood model ψ_i here is defined assuming perfect recall, or access to the full history h_i. It can be calculated as the product of the probability of each observation occurring against said opponent:

$$\psi_i(\pi_i \mid h_i) = \prod_{o_i \in h_i} \psi_i(\pi_i \mid o_i). \tag{6.6}$$

This can be efficiently calculated with a stateful policy that maintains the previous timesteps' belief and updates its posterior with the current observation's likelihood. We also consider a simpler likelihood that only depends on the current observation. Comparing these versions illuminates the predictive power of a single observation.

Table 6.1: **Comparison of Q-Mixing with differing opponent likelihood models.** The prior column denotes that the likelihood model can correct for updated prior knowledge. The likelihood column denotes whether the model updates the posterior given evidence during play.

Formula	Prior	Likelihood
$Q_{\pi_i}(h_i, a_i \mid \sigma_{-i}) \propto \sum_{\pi_{-i}} \bar{\sigma}_{-i}(\pi_{-i}) Q_{\pi_i}(h_i, a_i \mid \pi_{-i})$		
$Q_{\pi_i}(h_i, a_i \mid \sigma_{-i}) \propto \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) Q_{\pi_i}(h_i, a_i \mid \pi_{-i})$	✓	
$Q_{\pi_i}(h_i, a_i \mid \sigma_{-i}) \propto \sum_{\pi_{-i}} \psi_i(\pi_{-i} \mid h_i) Q_{\pi_i}(h_i, a_i \mid \pi_{-i})$		✓
$Q_{\pi_i}(h_i, a_i \mid \sigma_{-i}) \propto \sum_{\pi_{-i}} \psi_i(\pi_{-i} \mid h_i) \frac{\sigma_{-i}(\pi_{-i})}{\bar{\sigma}_{-i}(\pi_{-i})} Q_{\pi_i}(h_i, a_i \mid \pi_{-i})$	✓	✓

6.2 Baseline Opponent Classifiers

Before we can begin to understand the details of what constitutes a good method for maintaining an opponent identifier, we must first understand where we start to contextualize progress. Do we even need to perform opponent identification? If the tools we already have perform well then pursing this investigation may be a nonstarter.

In the simplest case, let's suppose that we do not need to explicitly consider the identity of the opponent. This amounts to selecting a best-response policy irrespective of information provided about the opponent's strategy. In Figure 6.1, we compare Q-Mixing: Prior directly against the best-response policies. We look at each method by evaluating their coverage across the opponent's strategy space in the RWS game. This is the same experimental methodology that we introduced in Chapter 5.

The best-responses to the opponent's policies are denoted BR(X), where X is the index of the opponent's policy in their strategy set. These baselines let us investigate the option of choosing the simplest opponent classifier: a classifier that outputs a constant value. These also allow us to see if any one particular best-response policy dominates the others. Such a dominant policy could serve as a sufficiently good policy against the mixed strategy.

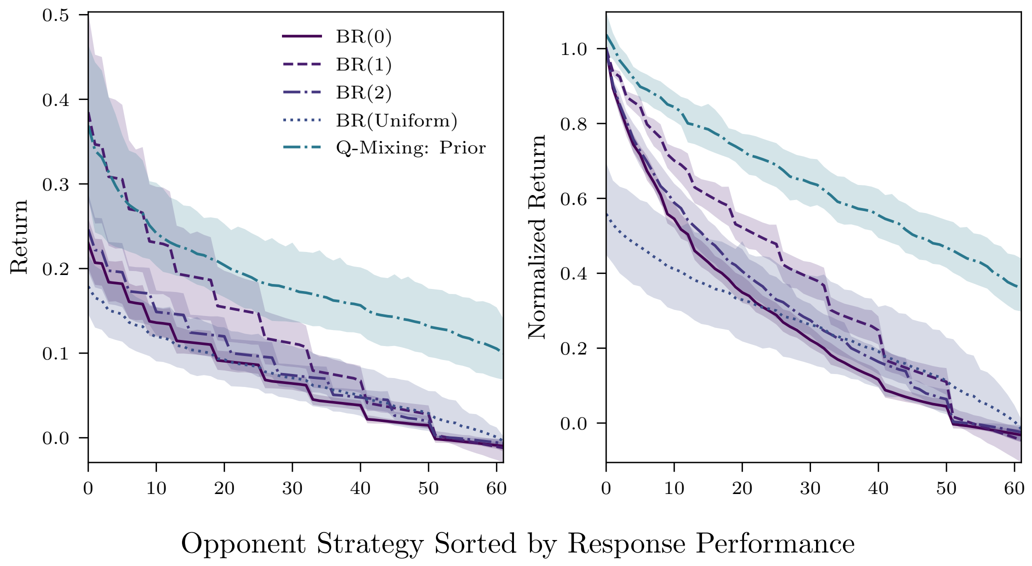


Figure 6.1: **Coverage of best-response policies.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy.

We also consider the best-response to the opponent's uniform mixed strategy, labelled BR(Uniform). BR(Uniform) implicitly should perform both opponent belief maintenance and best-response to perform well. Investigation of this baseline should offer some insights into how easily identifiable the opponent's policy is from the observation. If this is an easy task, then the implicit two-step response strategy of identification and then response is easy to perform at all game states. Therefore, a good BR(Uniform) should highly correlate its Q-values with the observation features corresponding to the opponent's identity.

In Figure 6.1, we can see that the best-responses to the opponent pure strategies generalize poorly. This can be seen in the right plot, where the the curves start out at 1.0, which denotes that the response policy was playing against the corresponding opponent pure strategy. Then when the opponent plays any mixed strategy, the performance quickly falls off. Recall, that the normalized return reports the proportion of the return received when compared to the return received by playing the true response policies. A decline in this chart suggests that the pure strategy response policies are unable to exploit the different opponent policies. This indicates that the opponent's strategy space is sufficiently diverse that we cannot rely on just a response to an opponent pure strategy.

In the same figure, we can also see that BR(Uniform) at its best performs at just over half its potential. A reasonable retort is to remark that this result indicates a failure in the approximate best-response oracle. It is true that using DRL as an approximate oracle method can result in variable performance to the whim of policy implementation and training hyperparameters. It is possible that under the perfect settings, DRL may produce a response policy to the mixed strategy that also exhibits generalization capabilities. Despite this true criticism, in practice, finding that setting may consume more resources than simply directly tackling the problem from a less idealized setting. In this work, we spent roughly equivalent resources attempting to optimize the learning settings for the responses to the pure- and mixed-strategies. Therefore, we argue that the results herewithin represent a roughly fair practical comparison of methods.

Q-Mixing: Prior outperforms both the pure- and mixed-strategy best-response policies. This should come at no surprise, because Q-Mixing: Prior can use the *additional* information of the opponent's strategy at evaluation time. The other methods offer no flexibility given the known change in opponent strategy. Therefore, there is a clear benefit to explicitly including a mechanism for opponent identification within a policy. Note, that in Figure 6.1 when interpreting outperformance, it is not the case that Q-Mixing: Prior is better than the specialized BRs against their respective opponents. The performance of each curve is sorted for each method separately, the strategies represented by each point on the x-axis often differ between methods.

6.3 Frequency-Based Classification

The result from the previous section indicate that strictly using best-responses may be insufficient. However, each of these respective best-responses performs well against their respective opponent. Q-Mixing's ability to fully utilize the component best-responses depends on its efficacy at identifying the opponent's policy. In this section we increase the complexity of our Q-Mixing policy by a single step. Instead of considering only a prior (fixed or adjusted to the correct opponent strategy), we investigate inclusion of evidence during play to inform the identity of the opponent. As the trained classifier we saw in Chapter 5 used a neural network, attempting to understand its success and failure modes presents itself as an entirely disparate research direction.

As a stopgap, we will now consider an *observation frequency* based opponent-policy likelihood. This likelihood model is based of a simple statistic, and will allow us to dive into what contributes to the success of Q-Mixing. It will weight the SRQVs by the relative frequency of the observation occurring against each opponent. We use the replay buffers $\mathbb{B}_i$ used to train the respective responses i as datasets to calculate the observation frequencies. More formally, we compute the weight of assigned to each response j for player i as:

$$w_j(o_i) = \frac{\sum_{\bar{o}_i \in \mathbb{B}(\text{BR}_i(\pi^j_{-i}))} [\bar{o}_i = o_i]_{\mathbb{1}}}{\sum_{\bar{o}_i \in \bigcup_k \mathbb{B}(\text{BR}_i(\pi^k_{-i}))} [\bar{o}_i = o_i]_{\mathbb{1}}}, \tag{6.7}$$

and *Q-Mixing: Observation Frequency* (Freq) follows from this formulation:

$$Q_i^{\text{Freq}}(o_i, a_i \mid \sigma_{-i}) = \begin{cases} \sum_{\pi_{-i}^j} w_j(o_i) \cdot Q_i(o_i, a_i \mid \pi_{-i}^j) & o \in \bigcup_k \mathbb{B}(\mathbf{BR}_i(\pi_{-i}^k)) \\ \sum_{\pi_{-i}^j} \frac{1}{|\Pi_{-i}|} \cdot Q_i(o_i, a_i \mid \pi_{-i}^j) & \text{Otherwise} \end{cases} . \tag{6.8}$$

We compare Q-Mixing: Freq with its prior-only versions of Q-Mixing and the learned classifier in Figure 6.2. Q-Mixing: Uniform Prior is another baseline introduced here, representing the performance of Q-Mixing when it is not given the opponent's strategy, but rather maintains a constant strategy (in this case, the uniform strategy) across all opponent strategies. Interestingly, Q-Mixing: Uniform Prior outperforms its opponent across all strategies. This outcome can be attributed to the tendency of value functions to overestimate values. As a result, the value functions associated with less likely opponents are typically suppressed. This phenomenon acts as an implicit form of opponent modeling, which arises due to the specific experimental setup chosen in this case. However, this observation may not hold true in general.

The result in Figure 6.2 indicates that the performance of Q-Mixing: Freq falls between Q-Mixing: Prior and Q-Mixing: Uniform Prior. This finding is initially concerning, as it suggests that the evidence does not aid in opponent identification. In fact, Q-Mixing: Freq performs only marginally better than Q-Mixing: Uniform Prior, which receives no information about the opponent's strategy.

Why doesn't the observational evidence lead to improvements in Q-Mixing? To answer this question, we need to examine the replay buffers underlying Q-Mixing: Freq's implementation. Each replay buffer contains 100,000 experiences and corresponding observations. From these experiences, the replay buffers have 18,854, 20,464, and 21,040 unique states, respectively. This implies that agents do revisit observations quite frequently.

However, the more critical question is how many of these unique observations co-occur between pairs of replay buffers? Co-occurrence indicates that the agent should be uncertain about their opponent's identity, and the relative frequency could provide valuable information. In Table 6.2, we present the co-occurrence frequencies. This table reveals that very few observations (less than 500) ever co-occur between any two replay buffers. This means that if an observation appears in a replay buffer, it is highly informative of the opponent's identity.

So far we have established that our frequency baseline fails to effectively use evidence during play to improve Q-Mixing. Next, we saw that if an observation is in one of the response policies' replay buffers then it is highly informative of the opponent's identity. This suggests an intuitive contradiction with the former result. The resolution to this conflict lies in the second case present in the observation-frequency formula (Equation 6.8). What happens when an observation *is not* present in any replay buffer?

Table 6.2: **Unique observations common across the responses' replay buffers.** The rows and columns represent the replay buffers of the approximate best-response policies' replay buffers. Each cell represents the count of the number of unique observations that occur in both replay buffers.

	0	1	2
0	18,854	341	237
1		20,464	424
2			21,040

We begin by simulating Q-Mixing: Freq against each respective opponent for 300 episodes. From this simulation results we looked at the proportion of observations encountered that did not occur in any replay buffers. We refer to this situation as a cache-miss. The percentage of cache-misses against each opponent is $87 \pm 1\%$, $69 \pm 3\%$, and $85 \pm 2\%$. Here in lies the problem with this baseline. Despite the replay buffers being largely informative, the information is never able to bear fruit, because the agent is mostly experiencing novel states. Thus, it behaves only slightly better than Q-Mixing: Uniform Prior, because upon a cache-miss it implements the Q-Mixing: Uniform Prior policy (the second case in Equation 6.8).

The previous result demonstrates that the observations used to construct our policy often differ from those experienced during evaluation. The key takeaway from this experiment is that this version of Q-Mixing: Freq does not take the past into account. Currently, the classifier receives the present observation and predicts the opponent's identity. This prediction is based on the probability of each opponent for the current observation, which can be used to create and maintain a belief about the opponent's identity throughout an episode.

Upon revisiting our observation-frequency baseline, we can readily identify the significant benefits of belief maintenance. In the initial version of Q-Mixing: Freq, the predicted opponent likelihood becomes the uniform distribution in the event of a cache miss. However, by maintaining a belief over time, the likelihood from the previous prediction is preserved in the case of a cache miss. This approach facilitates the accumulation of evidence across time, even when faced with new observations. From this point forward, we will focus on a version of Q-Mixing: Freq that incorporates belief maintenance over time.

Figure 6.3 displays the coverage curves for various Q-Mixing: Freq variants. These plots show that maintaining a belief significantly enhances the performance of Q-Mixing: Freq. These improvements can be attributed to the agent's ability to accumulate all evidence about their opponent. Additionally, when the agent encounters a new observation, they no longer have complete uncertainty about their opponent. Instead, their previous belief about their opponent remains. As most observations experienced were novel, this means that the agent can now historic evidence of their

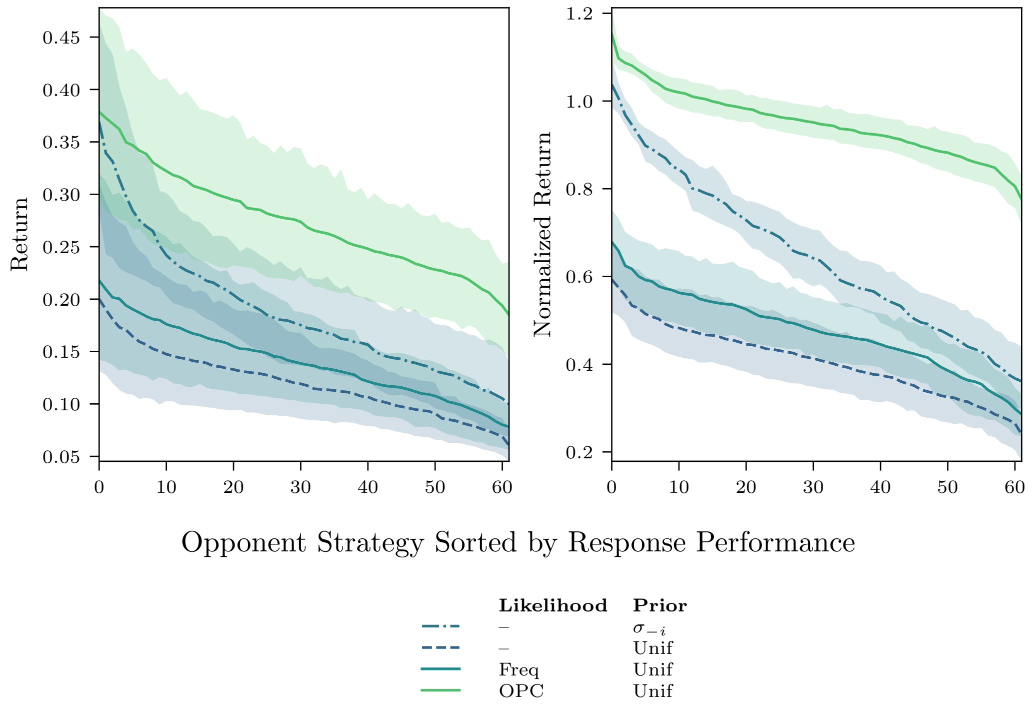


Figure 6.2: **Coverage of baseline variants of Q-Mixing.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy. This is the only figure that contains results for Q-Mixing: Freq that does not maintain a belief.

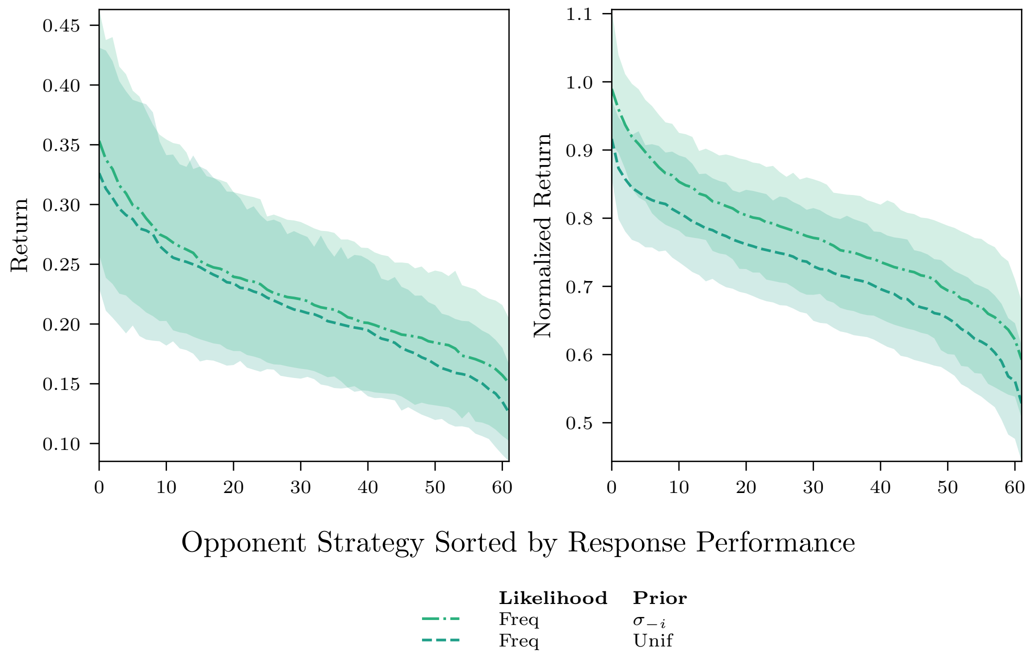


Figure 6.3: **Coverage of variants of Q-Mixing with a frequency-based likelihood.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy.

opponent.

Continuing from the Q-Mixing: Freq results, we can see that the prior plays a much smaller role in the performance of opponent identification. An informative prior provides a small increase in performance across the opponent's strategy space. The prior serves a minor role, because throughout an episode enough evidence about the opponent is collected to overwhelm the contribution from the prior. Still, a prior offers benefits early in an episode, by allowing an agent to deviate to a more exploitative response policy earlier. In practice, we will have access to the true opponent strategy; therefore, this result unsurprisingly suggests that it should be used as the prior.

6.4 Learned Classifier

Finally, we return to our learned opponent classifier. We provide both coverage curves for Q-Mixing: OPC in Figure 6.4 and opponent classification accuracy of all methods (at the end of the episode) in Table 6.3. The learned classifier further improves upon the frequency baseline, as shown in Figure 6.5. The previous investigation into the performance of the frequency base-

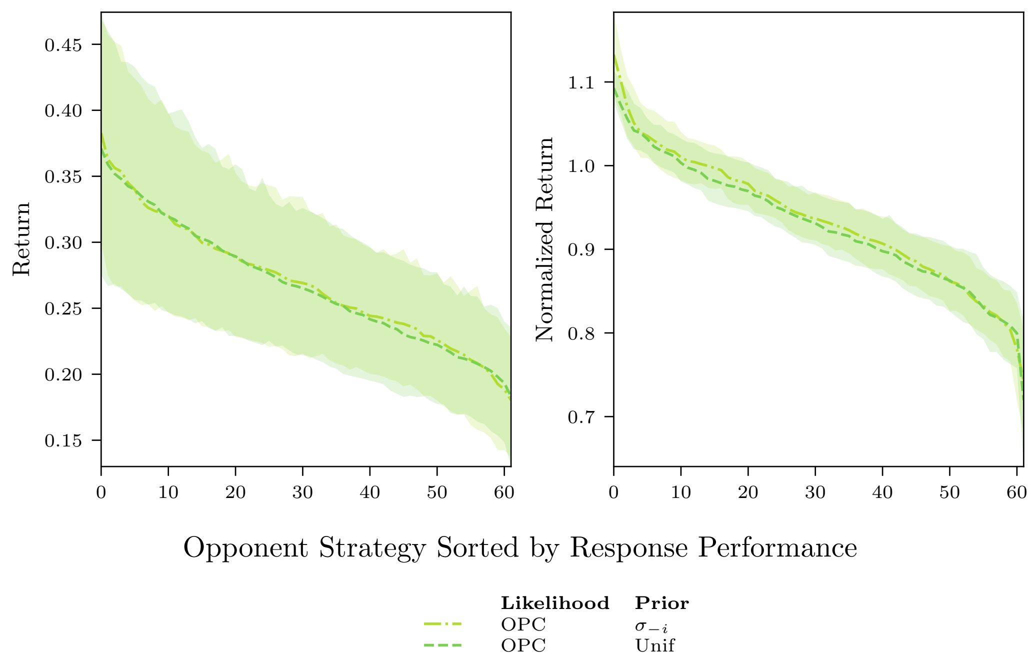


Figure 6.4: **Coverage of variants of Q-Mixing with a learned likelihood.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy.

line offers insights into the benefits gained from the learned version. With regards to extracting evidence of the opponent's identity: the OPC learns to extract features salient for opponent classification from its observations; whereas, the frequency baseline can only gather evidence from known observations contained in a replay buffer.

In summary, we compile all of the coverage curves analyzed in this section in Figure 6.5. The key take-away is that both the prior and evidence gained matter for successful belief calculation. The prior, is often of less importance, because it is typical to gain sufficient evidence during play to overwhelm the impact of the prior. Nevertheless, in games where the first few moves are crucial, the prior may prove fundamental to the method's success. A trained opponent-policy classifier may also learn to extract features that are predictive of the opponent's policy even in novel settings.

6.5 Future Research Directions

In this study we consider only a simple learned classifer for the OPC. Instead, more sophisticated methods for reasoning about the opponent's policy offers ample room for future improvements for

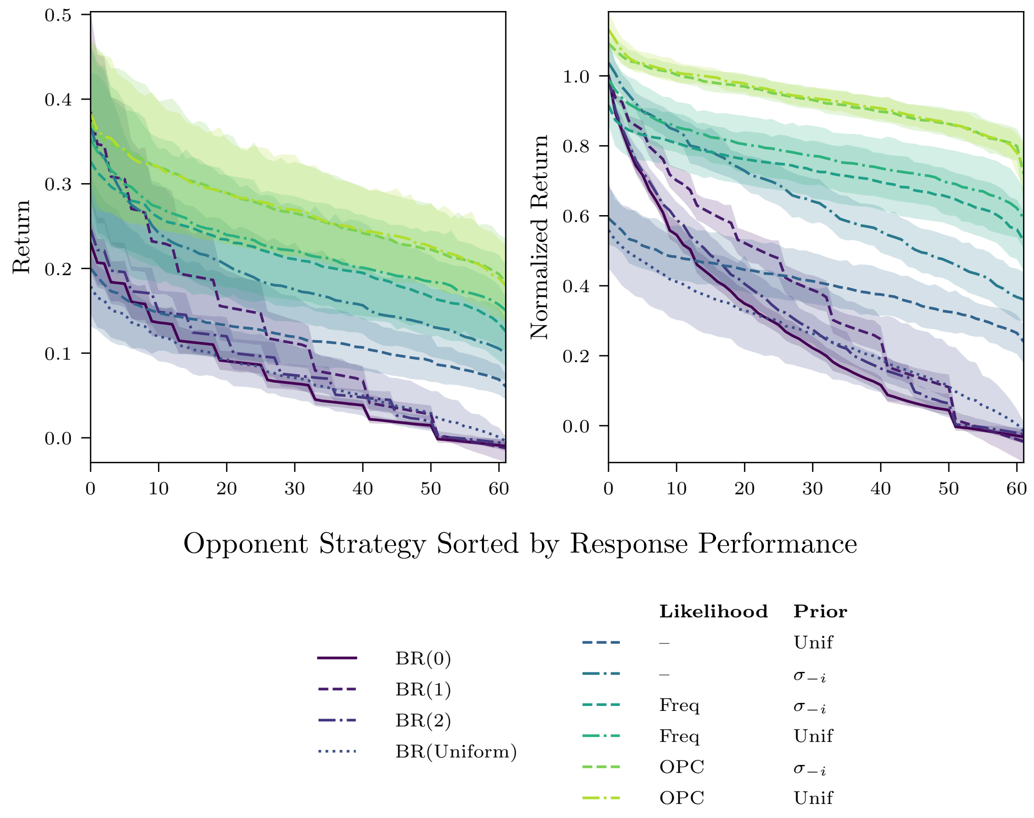


Figure 6.5: **Coverage of all methods on RWS.** The opponent strategies are sorted per-BR-method by the BR's return. Shaded region represents a 95% bootstrap confidence interval over five random seeds. The two methods are trained using the same simulation budget. The left plot, evaluates each method by their return. On the right plot, we instead normalize the return by the performance of the BR to each opponent policy.

Table 6.3: **Opponent classification accuracy at end of episode.** Accuracy is calculated over 100 episodes and intervals are calculated from an empirical bootstrap across 5 seeds. Accuracy is only calculated for the final timestep in an episode. Tie-breaking favors the smaller opponent index, resulting in high accuracy for Q-Mixing: Freq.

Likelihood	Prior	Opponent 0	Opponent 1	Opponent 2
Freq	Unif	0.67 ± 0.16	0.61 ± 0.10	0.63 ± 0.06
Freq	σ_{-i}	0.73 ± 0.09	0.69 ± 0.12	0.78 ± 0.05
OPC	Unif	0.89 ± 0.03	0.89 ± 0.02	0.90 ± 0.03
OPC	σ_{-i}	0.92 ± 0.04	0.93 ± 0.03	0.91 ± 0.03

Q-Mixing. A set of assumptions that can be made includes that *all* players have fixed strategy sets. Under these assumptions, agents could maintain more sophisticated beliefs about their opponents [Zheng et al., 2018], and extend this to recursive-reasoning procedures [Yang et al., 2019a,b, 2021b]. This line of work primarily focuses on other-player policy identification and presents a promising future direction for enhancing the quality of the OPC.

Another potential extension of the OPC is to explore alternative objectives. Rather than solely focusing on predicting the opponent, in safety-critical situations, an agent may want to consider an objective that accounts for inaccurate opponent predictions. The Restricted Nash Response embodies this measure by striking a balance between maximizing performance if the prediction is correct and maintaining reasonable performance if the prediction is inaccurate [Johanson et al., 2007].

While both of these research directions revolve around opponent-policy prediction, they address different problem statements. Most notably, these works do not consider varying the distribution of opponent policies as we have investigated in this work. As a result, adapting these methods to this distinct problem domain presents a fruitful opportunity for future research.

CHAPTER 7

Game Solving

Up to this point, we have focused on the issue of transferring responses across opponent strategies. Now, we turn our attention to investigating how these advances may reduce the cumulative cost of learning in game-solving algorithms. PSRO is one such algorithm for learning a solution to a in multiagent systems by interleaving empirical game analysis with DRL. At each iteration, DRL is invoked to train a best-response to a mixture of opponent policies. The learning problems faced across each iteration share a common structure. This common structure provides us with the opportunity to transfer knowledge acquired in previous iterations to assist in training subsequent policies. When consecutive best-response problems are compared only two changes may occur: inclusion of an additional policy in each opponent's strategy set, and a change in the distribution with which each opponent samples their policies. In this section, we introduce two variants of PSRO that exploit this common strategic structure to reduce the amount of simulation required during DRL training.

A particular challenge for the RL step in PSRO is that the learner must derive a response under uncertainty about opponent policies. The profile derived from the empirical game is generally a mixed-strategy profile, as in strategically complex environments randomization is often a necessary ingredient for equilibrium. The opponent draws from this mixture are unobserved, adding uncertainty to the multiagent environment. We address this challenge through variants of PSRO in which all RL is applied to environments where opponents play pure strategies. The proposed methods employ, but are not limited to, the machinery of *Q-Mixing* to facilitate operations on pure strategies instead of mixed strategies. We propose and evaluate two such methods, which work in qualitatively different ways: *Mixed-Oracles* learns separate BRs to each pure strategy in a mixture and combines the results from learning to approximate a BR to the mixture. *Mixed-Opponents* constructs a single pure opponent policy that represents an aggregate of the mixed strategy and learns a BR to this policy.

Our methods promise advantages beyond those of learning in a less stochastic environment. Mixed Oracles transfers learning across epochs, exploiting the Q-functions learned against a particular opponent policy in constructing policies for any other epoch where that opponent policy is encountered. Mixed Opponents applies directly over the joint opponent space, and so has the potential to scale beyond two-player games.

7.1 Mixed-Oracles

The first problem we address is that during BR calculation there is an opportunity for transferring previously learned information. In each epoch, each player learns a BR to a mixed profile of opponent policies. This mixture typically involves the newly added policies (one per player) for this epoch, but may also include many policies from previous epochs. Training in previous epochs already captured experience against those policies, so including them in further training may be redundant.

The *Mixed-Oracles* algorithm is a variant of PSRO for two-player games, with a modified BR oracle designed to transfer learning across epochs. This method works by learning and maintaining a collection of BRs to each opponent policy $\Lambda_i^e = \{\lambda_i^1, \dots, \lambda_i^e\}$, where λ_i^e is the BR to π_{-i}^{e-1}. During each epoch of Mixed-Oracles, a BR is learned for the single new opponent policy, rather than for the mixed opponent-profile generated by the MSS. A BR to the MSS-generated target mixture is then *constructed* from the collection of BR results for constituent policies in the mixture. Constructing the new policy is done through a general *TransferOracle* function that maps a set of policies and a distribution over the policies into a single policy

$$\text{TransferOracle} : \Pi \times \Delta(\Pi) \to \pi. \tag{7.1}$$

The resulting policy should approximately aggregate the behavior of the component policies.

By reusing learned behaviors from previous epochs, Mixed-Oracles allows us to focus training exclusively on new opponent policies. The key design choice is how to combine knowledge from the BRs to individual policies into a BR to any distribution of said policies. We provide a general description of Mixed-Oracles, where the TransferOracle method is abstract, as Algorithm 3.

Chapter 5 introduces *Q-Mixing* as an approach for constructing policies against any mixture of opponent strategies. This method utilizes Q-values learned against each individual opponent strategy, making it well-suited for supporting the desired transfer. As we observed in Chapter 4, direct policy transfer is not always feasible. Consequently, Q-Mixing's use of values makes it particularly appropriate for aiding Mixed-Oracles. Specifically, Q-Mixing calculates the average Q-values learned against each opponent policy π_{-i}, weighted by their likelihood in the opponent mixture σ_{-i}:

$$Q_i(o_i, a_i \mid \sigma_{-i}) = \sum_{\pi_{-i}} \psi_i(\pi_{-i} \mid o_i, \sigma_{-i}) Q_i(o_i, a_i \mid \pi_{-i}), \tag{7.2}$$

where ψ determines the relative likelihood of playing an opponent $\psi_i : \mathcal{O}_i \rightarrow \Delta(\Pi_{-i})$. In this study, we use Q-Mixing as our TransferOracle, where ψ is the prior over the opponent distribution as given by an MSS.

Algorithm 3: Mixed-Oracles

```
**Input:** Initial policy sets for all players $\Pi^0$
Simulate utilities $\tilde{U}^{\Pi^0}$ for each joint $\pi \in \Pi^0$
Initialize solutions $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$
Initialize pure strategy BRs $\Lambda_i^0 = \emptyset$
while epoch e in $\{1, 2, \ldots\}$ do
    # Best respond to each new opponent.
    for player $i \in [[n]]$ do
        for many episodes do
            Train $\lambda_i^e$ over $\tau \sim (\lambda_i^e, \pi_{-i}^{e-1})$
        $\Lambda_i^e = \Lambda_i^{e-1} \cup \{\lambda_i^e\}$
    # Generate new policies.
    for player $i \in [[n]]$ do
        $\pi_i^e \leftarrow \text{TransferOracle}(\Lambda_i^e, \sigma_{-i}^{*,e-1})$
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$
    Simulate missing entries in $\tilde{U}^{\Pi^e}$ from $\Pi^e$
    Compute a solution $\sigma^{*,e}$ from $\tilde{\Gamma}^e$
Output: Current solution $\sigma_i^{*,e}$ for player $i$.
```

7.1.1 Empirical Convergence of Mixed-Oracles

Does Mixed-Oracles yield a solution of similar quality to PSRO while using fewer simulation timesteps? We evaluate this question by comparing both methods cumulative simulation timesteps usage during game solving. We compare the methods on two distinct games: RWS and Gathering [Leibo et al., 2021], which are detailed in Chapter 2.

Figure 7.1 compares convergence speed, measured by their regret over time, of Mixed-Oracles and PSRO on the RWS game. Both Mixed-Oracles and PSRO converge to an equilibrium within the budgeted 3.5×10^8 timesteps. However, at 1.5×10^8 timesteps Mixed-Oracles converged to an equilibrium; whereas, PSRO has not. Mixed-Oracles converges in a similar number of epochs as PSRO seen on the left plot with both algorithms converging around six. However, Mixed-Oracles achieves this solution while requiring less usage of the environment simulator (measured in timesteps on the right plot). It is worth recalling here that both PSRO and Mixed-Oracles in this experiment is initialized with a strategy-set containing the three specialized policies (rock, paper, and scissors). This means that both algorithms contain many equilibria before performing

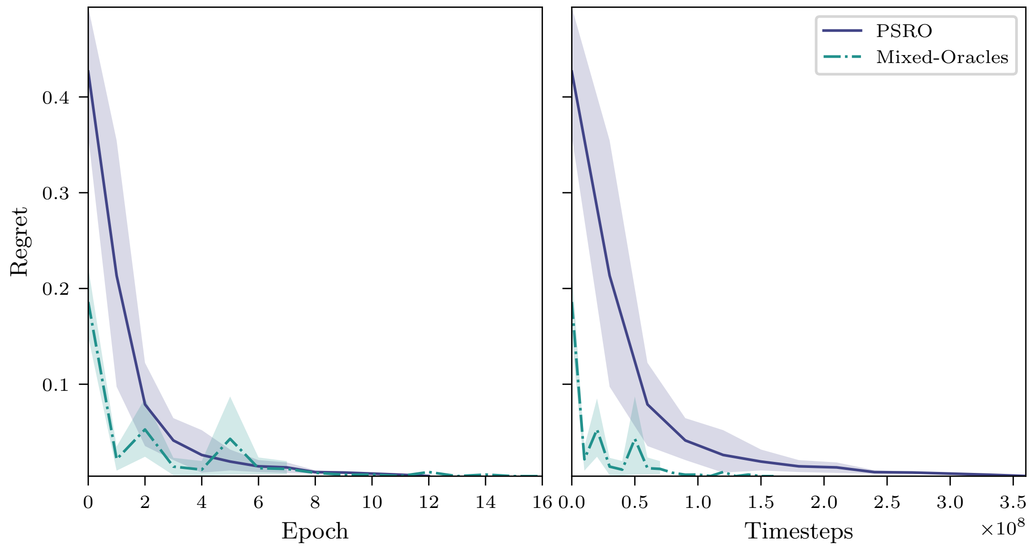


Figure 7.1: **Mixed-Oracles on the RWS game.**

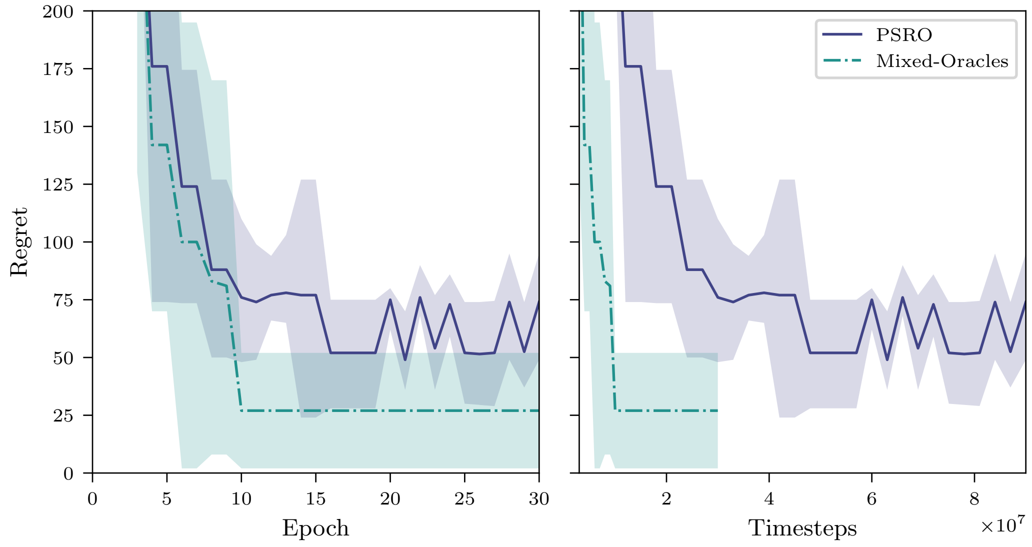


Figure 7.2: **Mixed-Oracles on the Gathering game.**

any strategy exploration. Therefore, we turn towards investigating other games where the strategy exploration step of PSRO has a greater influence on the algorithm's performance.

Figure 7.2 similarly compares both algorithms on the Gathering game with two-players and the small map. At epoch 10, Mixed-Oracles has converged to a solution with roughly 25 regret. At the same time, PSRO approximately triple the regret. By the end of PSRO's runtime it improves its performance to roughly 50 regret, double that of Mixed-Oracles. The reduction can be regret may be contributed to by two factors (a) reduction cost of each epoch (measured in simulated timesteps) allows more epochs to be run, and/or (b) noise introduced through Q-Mixing's approximation helps in exploring the game's strategy space. In this game, both algorithms appear to converge roughly around 10 epochs. This suggests that the later factor, noise-induced exploration, may have played a larger role in the overall algorithm's performance improvements.

7.2 Mixed-Opponents

We next examine PSRO's strategy exploration method that is defined by BR to the opponent profile generated by an MSS. This design choice was motivated by solving for Nash Equilibrium, where failure to add a beneficial deviation to the restricted strategy set indicated convergence. However, in an extensive form game it can take infeasibly many iterations to achieve convergence, meaning that in practice the theoretical convergence in the limit may not be helpful.

In other words, BR within a single iteration serves the short-term goal of checking for convergence presently, but may not serve the long-term goal of building a rich empirical game. This distinction is akin to the exploration-exploitation dilemma: we may solve a game faster by choosing not to respond to the current Nash equilibrium, if it leads to strategies that are useful in improving our empirical game. Exploration-focused objectives promise to reduce the number of iterations of PSRO, instead of reduce the number of timesteps of a single RL application.

In this section, we introduce Mixed-Opponents, a variant of PSRO that incorporates an exploration-focused objective. The key insight behind this objective is that each opponent policy is greedy, which leads to the suppression of potentially useful information already learned about non-greedy actions. For instance, many opponent policies may agree on a second-best action that is never played. We propose Mixed-Opponents, which employs the transfer learning methods developed in this work to help us explore this direction. Similar to Mixed-Oracles, Mixed-Opponents maintains the advantage of responding to specific opponent policies, resulting in less stochastic learning signals and cheaper response learning. To begin, we will present Mixed-Opponents through a constructed example.

7.2.1 Constructed Example

In this example, we consider a hypothetical run of PSRO to solve Rock-Paper-Scissors (RPS) with Nash equilibrium as its solution concept. We will focus on Player 1's learning, so will assume that Player 2 will learn exact Q-values against Player 1's meta-strategy. However we will consider Player 1 to only produce approximate best responses. This is to model approximate reinforcement learning such as observed in experiments on the abstracted RPS game, running-with-scissors For simplicity, we will consider each player adding a strategy to the empirical game in turn, rather than simultaneously.

The below table shows a possible outcome of two iterations of PSRO.

$$\begin{array}{rl} & \text{R}\quad \text{P}\quad \text{S} \\ \pi_1^1 &= (0.0, 0.3, 0.7) \\ \pi_1^2 &= (0.4, 0.6, 0.0) \end{array} \qquad \begin{array}{rl} & \text{R}\quad\ \text{P}\quad\ \text{S} \\ Q_2^1 &= (0.4, -0.7, 0.3) \\ Q_2^2 &= (-0.6, 0.4, 0.2) \end{array}$$

The run begins by Player 1 playing an arbitrary initial strategy π_1^1, which consists of mostly playing S. This play induced a high value for Player 2's R action, denoted in Q_2^1. In turn, Player 1 approximately best-responds to R, by playing mostly P, and never playing S, in π_1^2. Player 1's meta-strategy now plays pure π_1^2, against which both P and S score well, as shown by the values of Q_2^2.

These policies construct an empirical game, which closely resembles the matching pennies game, with the following payoffs:

		Player 2	
		π_2^1	π_2^2
Player 1	π_1^1	$-0.4, 0.4$	$0.7, -0.7$
	π_1^2	$0.6, -0.6$	$-0.4, 0.4$

This game is then solved by the MSS resulting in the following solution:

$$\sigma_1^2 = (0.47\,\pi_1^1,\ 0.52\,\pi_1^2)$$
$$\sigma_2^2 = (0.52\,\pi_2^1,\ 0.48\,\pi_2^2).$$

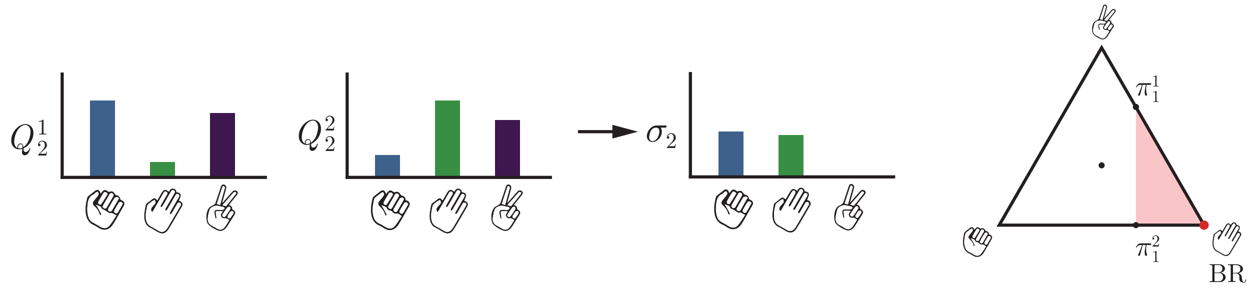
Because Player 2's two policies play only R and P respectively, we can rewrite Player 2's meta-strategy σ_2^2 in terms of the primitive actions in the full game $(0.52\,\text{R},\ 0.48\,\text{P},\ 0.0\,\text{S})$. If, as in PSRO, Player 1 were then to add their ABR to this meta-strategy, which is P, then their strategy set does not contain the Nash equilibrium of the game. The problem is that there are already strategies in the empirical game that can defeat both of Player 2's strategies, so the new strategy is not very useful. This can be detected by inspecting Player 2's Q-functions: Q_2^1 has a very low valuation of P, and Q_2^2 has a very low valuation of R. This phenomena is illustrated in Figure 7.3a. Player 1 could instead respond to S, which is moderately highly valued by both opponent Q functions, as it is more relevant. To detect this we can mix the opponent strategies through their action-value estimates, rather than their action distributions. This results in the following Q-values: $Q_2^{\text{Mix}} = (0.46, 0.414, 0.626)$. The ABR to this is R, which more usefully extends Player 1's strategy set to include the Nash equilibrium of this game. The two different approaches are illustrated in Figure 7.3b.

7.2.2 Mixed-Opponents

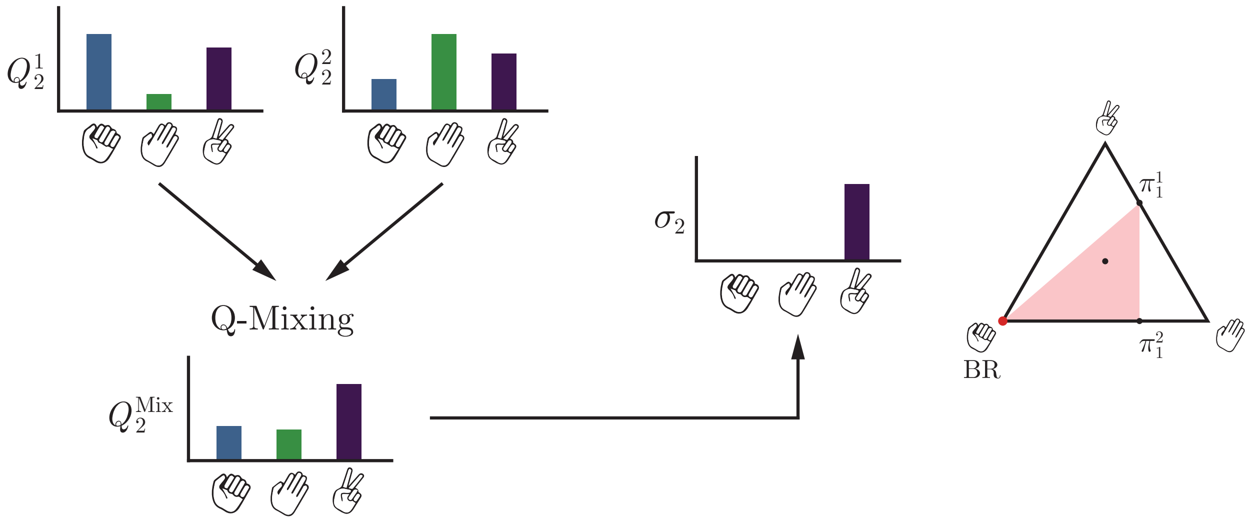
This example motivates our second algorithm: Mixed-Opponents. This method also employs a combination method, but instead of combining results from training against previously encountered opponents, it combines the strategies of the opponent mixture themselves to construct a single new opponent policy as a target for training. We refer to the method for generating a new opponent policy from a mixture of opponents the OpponentOracle, and it has the same functional form as the TransferOracle. The generalized Mixed-Opponent algorithm is shown in Algorithm 4. We employ Q-Mixing (Equation 7.2) as our OpponentOracle. In contrast to Mixed-Oracles, which uses Q-Mixing to transfer Q-values across epochs, here we apply it to average Q-values to define a variant training objective.

Algorithm 4: Mixed-Opponents

```
Input: Initial policies for all players $\Pi^0$
Simulate $\tilde{U}^{\Pi^0}$ for each joint $\pi \in \Pi^0$
Initialize solutions $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$
while epoch e in $\{1, 2, \ldots\}$ do
    for player $i \in [[n]]$ do
        $\pi_{-i} \leftarrow \text{OpponentOracle}(\Pi_{-i}^{e-1}, \sigma_{-i}^{*,e-1})$
        for many episodes do
            Train $\pi_i^e$ over $\tau \sim (\pi_i^e, \pi_{-i})$
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$
    Simulate missing entries in $\tilde{U}^{\Pi^e}$
    Compute a solution $\sigma^{*,e}$ from $\tilde{\Gamma}^e$
Output: Solution $\sigma_i^{*,e}$ for player $i$.
```



(a) PSRO with Nash equilibrium as a solution concept. The two opponents' Q-functions inform greedy policies that each play rock and paper, respectively. When the meta-strategy for Player 2 is $\sigma_2 = (0.52\,\pi_2^1,\ 0.48\,\pi_2^2)$, Player 1 will add paper as a BR.



(b) Mixed-Opponents first mixes the opponent policies by their Q-values (in this example using the Q-Mixing algorithm). The BR to the mixed opponent is rock, and its inclusion expands the strategy space to include the Nash equilibrium of RPS (middle dot).

Figure 7.3: **Empirical game expansion resulting from different strategy exploration methods.**

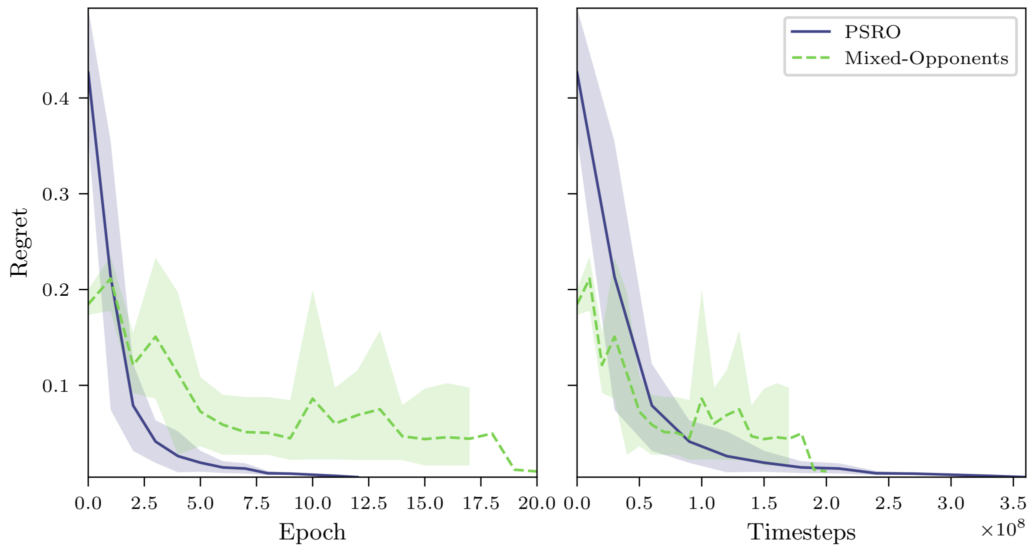


Figure 7.4: **Mixed-Opponents on the RWS game.**

7.2.3 Empirical Convergence of Mixed-Opponents

The first research question we address is: does Mixed-Opponents lead to a solution of similar quality compared to PSRO while utilizing fewer simulation timesteps? To answer this question, we follow the same experimental procedure used for the Mixed-Oracles experiment in the previous section.

Figure 7.5 compares the convergence speed of both algorithms on the Gathering-Open game. After 25 epochs, Mixed-Opponents has discovered a solution with approximately 50% less regret. However when both algorithms are compared via timesteps, at 5×10^7 timesteps, Mixed-Opponents has found a nearly no regret solution; whereas, PSRO has approximately 50 regret.

Mixed-Opponent's performance on the RWS game is shown in Figure 7.4. In this result, we can see that PSRO behaves as expected: decreasing regret through epochs and converging to a low-regret solution. On the other hand, Mixed-Opponents, behaves erratically and does not show convergence. What appears as a complete failure of the method, provides the practitioner some valuable insights into the utility gained through differing strategy exploration methods. Recall from the discussion on Mixed-Oracles applied to RWS that Mixed-Oracles was initialized to contain several equilibria. The same consideration also applies to Mixed-Opponents; meaning that Mixed-Opponents need not explore the strategy space of the game to discover equilibria. This means that we are evaluating Mixed-Opponents in a setting which contradicts its motivation. Naturally, any gains Mixed-Opponents offers by discovering new strategies would not be advantageous. This also highlights a downside of Mixed-Opponents: it may fail to *exploit* the discovered strategy space. Instead of discovering new strategies, this setting requires exploiting the discovered strategy space

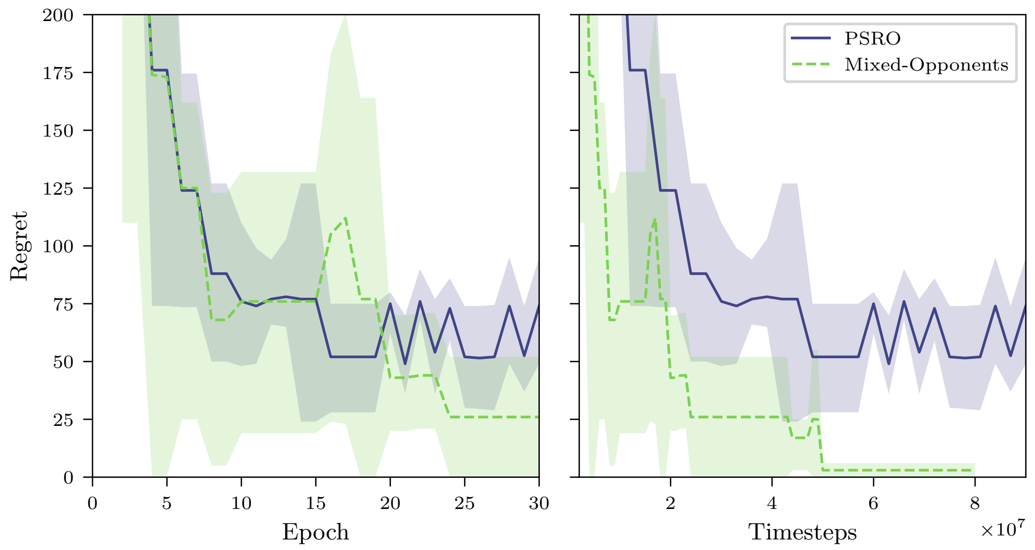


Figure 7.5: **Mixed-Opponents on the Gathering game.**

to compute an accurate equilibrium response policy. We discuss this in more detail in Section 7.4.

7.2.4 Many-Player Games

An advantage of Mixed-Opponents over Mixed-Oracles is that it can be applied to games with more than two players. It is natural to then ask if the trends we observed in the previous experiment, on two-player games, extend to many-player games? We repeat the previous Mixed-Opponent analysis on the Gathering game; however, we will now look a three-player version of the game on the “open” map (maps define spawn points and orchard configurations). We limit each profile in the empirical game to three simulations, to handle the combinatorial explosion of profiles. Figure 7.6 shows our results where Mixed-Opponents finds a similar quality solution to PSRO in half of the time.

7.3 Hyperparameter Selection Ablation

In the previous experiments PSRO used a separate set of hyperparameters from the proposed algorithms. These two sets of hyperparameters were specialized for low- and high-variance outcomes of state induced by facing pure- and mixed-strategy opponents respectively. This was motivated by the assumption that lower variance would require less training. This raises the question: does the differing hyperparameters explain the performance gap between the algorithms?

In this section, we question that assumption and ask: do Mixed-Oracles and Mixed-Opponents

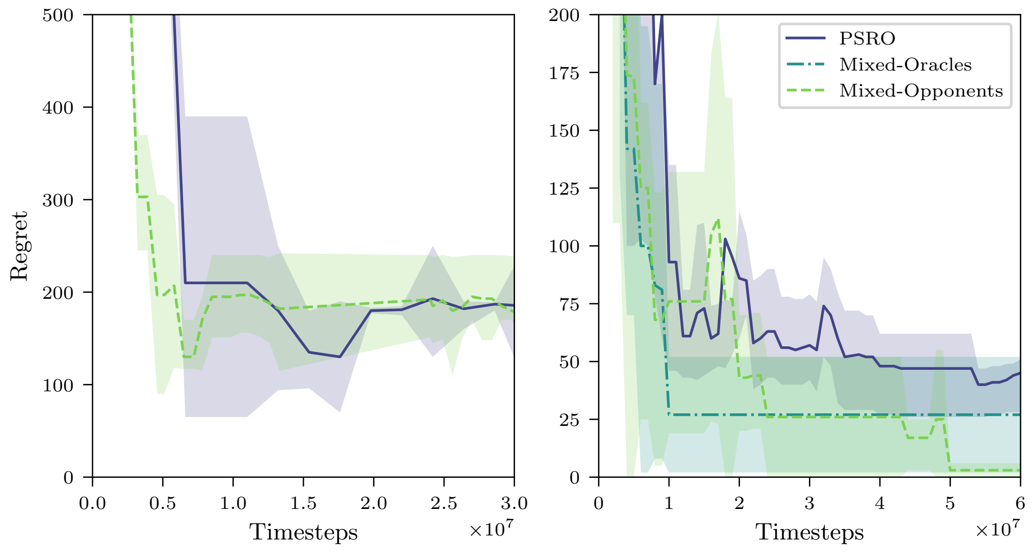


Figure 7.6: **Extensions beyond two players and using shared hyperparameters.** (Left) Mixed-Opponents evaluated on the Gathering-Open game with 3 players. (Right) Comparison of algorithms when set to the same DRL hyperparameters.

perform at least as well as PSRO when given the same DRL hyperparameters? To answer this question we run all three algorithms with the same set of hyperparameters, forcing all to adopt the same simulation budget.

We report results for the Gathering-Small game in Figure 7.6. The trends observed previously reoccur: Mixed-Oracles and Mixed-Opponents find solutions at least as good as PSRO after 6×10^7 timesteps. Moreover, by 2.5×10^7 timesteps both Mixed-Oracles and Mixed-Opponents have converged to a regret of approximately 25, while PSRO has a regret of roughly 50. These results suggest that our hyperparameter selection methodology does not explain the results from the preceding experiments.

7.4 Strategy Exploration-Exploitation Dilemma

DO uses response to Nash as a way to guarantee theoretical convergence in the limit. Specifically, if a new best-response strategy cannot be constructed to add to the empirical game for any player then a Nash solution is found. PSRO with Nash as a solution adopts this guarantee by inheriting the same algorithmic structure as DO. However, unlike DO, PSRO is applied to games where we cannot reasonably run the algorithm long enough to realize convergence. Instead of guaranteeing that each new strategy serves the additional role of a convergence check, we can also choose to select strategies that reduce the time till convergence. This trade-off is analogous to the exploration-exploitation dilemma in single-agent RL, where now it may be advantageous to first *explore* and add diverse strategies, then *exploit* and attempt to solve the game.

Rectified Nash PSRO is an example of a exploration method that encourages agents to "amplify their strengths and ignore their weaknesses," [Balduzzi et al., 2019]. This leads to the inclusion of generally weak agents that have diverse niches. The rectified Nash objective focuses on increasing the effective diversity of the population, and does not directly optimize towards solving the game. While response to the Nash PSRO is a an exploitative objective, where it assumes that all strategic cycles in the game are included in the empirical game.

We posit that Mixed-Oracles is similarly an exploitative objective, and that Mixed-Opponents is an exploration objective. Mixed-Oracles includes responses to the MSS's solution but introduces additional approximation errors by purifying pure-strategy best-response policies. Mixed-Opponents, on the other hand, attempts to add a more diverse policy to the population; following the intuition that non-optimal actions may contain interesting strategic dimensions in aggregate. An open question is characterizing how Mixed-Opponents impacts the quality of the empirical game. No algorithm so far acts as a panacea alone; however, mixing the insights gained from all of them together may offer a path towards the remedy.

7.5 Conclusion: Strategic Knowledge Transfer

In this part of the dissertation I investigated how transferring knowledge about previously encountered opponents can generalize response knowledge across opponent strategies and improve the efficiency of game-solving algorithms. The story began by introducing a class of problems and characterizing them as strategic knowledge transfer problems. These problems address the use of strategic knowledge accrued while learning response policies in one context toward deriving a response policy for a new strategic context.

We investigate one such problem: the opponent mixture transfer problem. In this problem, we assume responses to each opponent policy and access to the strategic mixture that the opponent is playing. First, we show how a general solution to this problem cannot be constructed without making additional assumptions about the policy's implementation. Then, we introduce *Q-Mixing*, an algorithm that solves the problem under the assumption that all response policies are value-based. Q-Mixing transfers response knowledge across any distribution of known opponents by appropriately weighting the responses' Q-values. We introduced exact methods for Q-Mixing, as well as approximate versions which we empirically demonstrate offer more practical solutions.

Key to the success of Q-Mixing, and solutions to the opponent mixture transfer problem in general, is the maintenance of a belief in the opponent's policy. Belief in the opponent's identity informs the select a suitable response behavior. We performed an in-depth analysis to tease apart what factors contribute to successful belief maintenance. In the games we tested, we saw that a opponent-policy classifier, trained using the replay buffers from the pure strategy response policies, served as an effective opponent-policy likelihood model. Moreover, the belief may be greatly improved by maintaining a posterior likelihood that is repeatedly updated at each observation.

Finally, we turned to the use of transfer learning to reduce the computational cost of iterative game-solving algorithms. We introduced two algorithms that share a common theme of modifying the best-response objective from responding to mixed-strategy opponents to responding to pure-strategy opponents. Responding to a pure strategy rather than a mixture eliminates the opponent sampling process, and thus reduces the variance in experiences during training.

The first algorithm, Mixed-Oracles, trains response policies to each policy in the population. A response-policy to a mixed strategy is then constructed by combining the individual pure-strategy response policies, using Q-Mixing. The second algorithm, Mixed-Opponents, transforms the mixed-strategy response target for PSRO into a pure strategy representing the mixture. It does so by combining the Q-values[1] of the opponent policies supported in the original target mixture, generating a novel policy that captures elements of the previous opponents. Both algorithms reuse strategic knowledge from previous PSRO iterations: the Q-values derived in training BRs. This reuse saves cumulative training time in PSRO, as does the variance reduction associated with responding to pure strategies noted above.

Mixed-Opponents also highlights the potential for novel *strategy discovery* as part of a *strategy exploration* approach in PSRO. As in single-agent RL, introduction of new strategy candidates for game solving must balance consideration of diverse possibilities (exploration) with fine-tuning of known effective solutions (exploitation).

These two algorithms are instances of learning-based game-solving algorithms that leverage *strategic knowledge transfer*. Their performance serves as evidence to the thesis of this dissertation that methods of transfer learning can reduce the cost of game solving. Future research should explore additional perspectives and extensions that could maximize the benefits derived from strategic knowledge transfer. With these points addressed, we conclude our discussion on strategic knowledge and shift our focus to the so far unnamed player: nature.

[1]Mixed-Opponents, like Mixed-Oracles uses Q-Mixing for the combination. Other approaches are possible, and should be investigated in future research.

Part III

World Knowledge Transfer

CHAPTER 8

Co-Learning Empirical Games & World Models

We seldom realize, for example, that our most private thoughts and emotions are not actually our own. For we think in terms of languages and images which we did not invent, but which were given to us by our society.

— Alan Watts

World knowledge refers to the understanding of game attributes that remain constant despite changes in players' strategies. This knowledge captures the underlying nature of the game, including its dynamics and reward signal. A complete knowledge of the world requires an agent to have experienced all possible transitions. Instead, an agent experiences only a limited view of the world that is shaped by the strategies of its inhabitants. This distinction differentiates our approach to world knowledge from that of strategic knowledge. While instances of strategic knowledge can be isolated, world knowledge must be continually refined as changing strategies reveal new facets of it.

Serendipitously, the constructing an empirical game presents us with just such a series of changing strategies. In this part of the dissertation, I explore how we can take advantage of this fact to learn, refine, and transfer world knowledge in conjunction with the construction of an empirical game. I represent world knowledge as a world model, which is a learned predictor of successor observations and rewards.

From the construction above it is clear that building an empirical game can benefit a world model. However, we will also see that co-learning both models can enhance both model's effectiveness. World models predict successor states and rewards given a game's current state and action(s). However, their performance depends on coverage of their training data, which is limited by the range of strategies considered during learning. Empirical games can inform training of world models by suggesting a diverse set of salient strategies, based on game-theoretic reasoning [Wellman, 2006]. These strategies can expose the world model to a broader range of relevant dynamics. Moreover, as empirical games are estimated through simulation of strategy profiles, this same simulation data can be reused as training data for the world model.

However, the strategic diversity offered by empirical games does carry a cost. As we have observed, each ABR calculation conducted by PSRO involves significant computational expense. World models, by facilitating the transfer of world knowledge, may help alleviate this cost through planning. Planning enables an agent to substitute real-world learning with learning in the world model, thereby potentially reducing the computational burden.

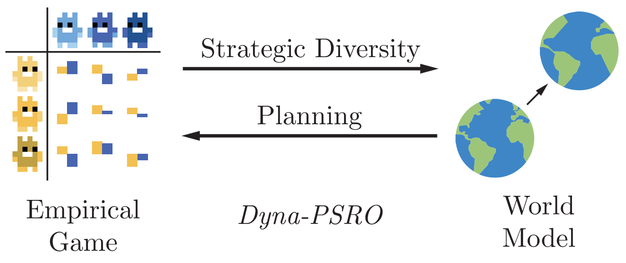


Figure 8.1: **Dyna-PSRO co-learns a world model and empirical game.** Empirical games offer world models strategically diverse game dynamics. World models offer empirical games more efficient strategy discovery through planning.

We investigate the mutual benefits of co-learning a world model and an empirical game by first verifying the potential contributions of each component independently. We then show how to realize the combined effects in a new algorithm, *Dyna-PSRO*, that co-learns a world model and an empirical game (illustrated in Figure 8.1). Dyna-PSRO extends PSRO to learn a world model concurrently with empirical game expansion, and applies this world model to reduce the computational cost of computing new policies. This is implemented by a Dyna-based reinforcement learner [Sutton, 1990, 1991] that integrates planning, acting, and learning in parallel. Dyna-PSRO is evaluated against PSRO on a collection of partially observable general-sum games. In our experiments, Dyna-PSRO found lower-regret solutions while requiring substantially fewer cumulative experiences.

8.1 Related Work

Previous research intersecting MARL and MBRL has primarily focused on modeling the opponent, particularly in scenarios where the opponent is fixed and well-defined. Within specific game subclasses, like cooperative games and two-player zero-sum games, it has been theoretically shown that opponent modeling reduces the sample complexity of RL [Tian et al., 2019, Zhang et al., 2020]. Opponent models can either explicitly [Mealing and Shapiro, 2015, Foerster et al., 2018a, Lockett et al., 2007, Ganzfried and Sandholm, 2011] or implicitly [Bard et al., 2013, Indarjo, 2019] model the behavior of the opponent. Additionally, these models can either construct a single model of opponent behavior, model others as onself [Raileanu et al., 2018], or learn a set of models [Collins, 2007, He et al., 2016, Shen and How, 2021]. While opponent modeling details are beyond the scope of this study, readers can refer to Albrecht & Stone's survey [Albrecht and Stone, 2018] for a comprehensive review on this subject. Instead, we consider the case where the learner has explicit access to the opponent's policy during training, as is the case in empirical-game building. A natural example is that of Self-Play, where all agents play the same policy; therefore, a world model can be learned used to evaluate the quality of actions with Monte-Carlo Tree Search [Silver et al., 2016, 2017a, Tesauro, 1995, Schrittwieser et al., 2020]. Li et al. [2023] expands on this by building a population of candidate opponent policies through PSRO to augment the search procedure. Krupnik et al. [2020] demonstrated that a generative world model could be useful in multi-step opponent-action prediction. Sun et al. [2019] examined modeling stateful game dynamics from observations when the agents' policies are fixed. Chockingam et al. [2018] explored learning world models for homogeneous agents with a centralized controller in a cooperative game. World models may also be shared by independent reinforcement learners in cooperative games [Willemsen et al., 2021, Zhang et al., 2022].

CHAPTER 9

World Models & Strategic Diversity

We begin by specifying exactly what we mean by world model. This requires defining some primitive elements. Recall that $t \in \mathcal{T}$ denotes the time in the real game, with $s^t \in \mathcal{S}$ the *information state* and $h^t \in \mathcal{H}$ the *game state* at time t. Note that this a change in definition of s, which was previously used to refer to Markov states, and h, which was previously used to refer to observation histories. We can refine our notion of information state $s^t \equiv (m^{\pi,t}, o^t)$ as being composed of the *agent's memory* $m^\pi \in \mathcal{M}^\pi$, or recurrent state, and the current observation $o \in \mathcal{O}$. Recall also that the *transition dynamics* $p : \mathcal{H} \times \boldsymbol{\mathcal{A}} \rightarrow \Delta(\mathcal{H}) \times \Delta(\boldsymbol{\mathcal{R}})$ define the game state update and reward signal.

An *agent world model* w represents dynamics in terms of information available to the agent. Specifically, w maps information states and actions to observations and rewards, $w : \mathcal{M}^w \times \boldsymbol{\mathcal{O}} \times \boldsymbol{\mathcal{A}} \rightarrow \boldsymbol{\mathcal{O}} \times \boldsymbol{\mathcal{R}}$, where $m^w \in \mathcal{M}^w$ is the *world model's memory*, or recurrent state. For simplicity, in this work, we assume the agent learns and uses a deterministic world model, irrespective of stochasticity that may be present in the true game.

A world model is trained to predict successor observations and rewards, from the current observations and actions, using a supervised learning signal. Ideally, the training data would cover all possible transitions. This is not feasible, so instead draws are conventionally taken from a dataset generated from play of a *behavioral strategy*. Performance of the world model is then measured against a *target strategy*. Differences between the behavioral and target strategies present challenges in learning an effective world model.

If the target strategy were known, we could readily construct the ideal training data for the world model. However the target is generally not known at the outset; indeed determining this target is the ultimate purpose of empirical game reasoning. The evolving empirical game essentially reflects a search for the target. Serendipitously, construction of this empirical game entails generation of data that captures elements of likely targets. This data can be reused for world model training without incurring any additional data collection cost.

9.1 Strategic Diversity

We call the probability of drawing a state-action pair $\boldsymbol{s},\ \boldsymbol{a}$ under a joint strategy $\hat{\boldsymbol{\sigma}}$ its *reach probability* $\eta^{\hat{\boldsymbol{\sigma}}}$. From this, we define *strategic diversity* as the distribution induced from reach probabilities. These terms allow us to observe two challenges for learning world models.

First, the diversity of the behavioral strategy *cover* the target strategy's diversity:

$$\eta^{\boldsymbol{\sigma}^*}(\boldsymbol{s}, \boldsymbol{a}) \rightarrow \eta^{\boldsymbol{\sigma}}(\boldsymbol{s}, \boldsymbol{a}), \tag{9.1}$$

which says that if there is support under the target strategy their must be support under the behavioral strategy. Otherwise, transitions will be absent from the training data. As an aside, it is possible to construct a weaker claim for coverage. This is done through making additional assumptions about the generalization capacity of a world model across transitions. For example, if transitions are drawn from two discrete latent variables, unseen combinations of these variables may be generalized if the individual values are known. However, generalization cannot be generally guaranteed, so we consider coverage.

The second challenge is that the *closer* the diversities are, the more accurate the learning objective will be. In other words, we want

$$\eta^{\boldsymbol{\sigma}^*}(\boldsymbol{s}, \boldsymbol{a}) \approx \eta^{\boldsymbol{\sigma}}(\boldsymbol{s}, \boldsymbol{a}). \tag{9.2}$$

If closeness is not ensured, the learning signal may be arbitrarily noisy, thus hampering learning the of crucial dynamics knowledge. An example of the issue of closeness can be seen in the "noisy TV problem" [Burda et al., 2019]. This exploration problem poses that novelty-seeking agents may be stuck forever watching the ever new TV static, and not experiencing practical novelty. In the same vein, if a world model is trained almost entirely on "noisy TV"-like experiences it may never learn. Therefore, we should strive to correct the distribution of experiences to be informed by a target strategy.

By design, empirical-game building algorithms offer a means to construct the target world model objective. These algorithms require the specification of a solution concept that serves the dual roll as the target strategy for a world model. Then through an iterative process, the empirical-game produces strategies that progressively approach the target strategy. This in turn, means that the process generates transitions that approach the target world model objective.

9.2 Experimental Setup

We evaluate the claims of independent co-learning benefits within the commons game Harvest: RGB, which is detailed in Chapter 2. To test the effects of strategic diversity specifically, we train a suite of world models that differ in the diversity of their training data. The datasets are constructed from the play of three policies: a random baseline policy, and two PSRO-generated policies. The PSRO policies were arbitrarily sampled from an approximate solution produced by a run of PSRO. We sampled an additional policy from PSRO for evaluating the generalization capacity of the world models. These policies are then subsampled and used to train seven world models. The world models are referred to by icons ▦ that depict the symmetric strategy profiles used to train them in the normal-form. Strategy profiles included in the training data of the world models are shaded black. For instance, the first (random) policy ▦, or the first and third policies ▦. Each world model's dataset contains 1 million total transitions, collected uniformly from each distinct strategy profile (symmetric profiles are not re-sampled). The world models are then evaluated on accuracy and recall for their predictions of both observation and reward for both players. The world models are optimized with a weighted-average cross-entropy objective.

9.2.1 Action-Conditioned Scheduled Sampling

As noted by Talvitie [2014], rolling out trajectories with an imperfect model tends to compound errors in prediction. Their work suggests training a Markovian world model with previous predictions (referred to as "hallucinated replay"), to train the model to correct errors. For stateful world models, as studied in this work, it has been demonstrated that curricula of n-step future predictions can train an effective world model [Michalski et al., 2014, Oh et al., 2015, Chiappa et al., 2017]. However, that body of work was focused on single-agent systems. Therefore, they benefited from a more stable data distribution for training when compared to a multiagent system. As a result, these fixed curricula can fail when transitioning them to multiagent systems.

Algorithm 5: Action-Conditioned Scheduled Sampling

```
$m \leftarrow$ Initial recurrent state
for $t \in T$ do
    $\boldsymbol{o} \leftarrow \boldsymbol{o}^t$ if Unif$[0,1] < \epsilon(t)$ else $\hat{\boldsymbol{o}}^t$
    $\hat{\boldsymbol{o}}^{t+1}, \hat{\boldsymbol{r}}^{t+1}, m \leftarrow w(\boldsymbol{o}, \boldsymbol{a}^t, m)$
Output: Predicted trajectory $(\hat{\boldsymbol{o}}^{0:T}, \hat{\boldsymbol{r}}^{0:T})$
```

Instead, this work adapts the scheduled sampling algorithm as a stochastic curricula, which will allow both short- and long-term predictions throughout the course of training [Bengio et al., 2015]. Scheduled sampling is an algorithm for training auto-regressive sequence prediction models where at each predictive step during training the model input is sampled from either the previous prediction or the ground truth. Adapting this algorithm for world model rollouts requires biasing each predictive step with the true actions while sampling between the predicted successor observation and the true successor observation. Therefore, the predictions will always be biased on true actions, but must learn to handle model-predicted observation. The sampling follows a schedule $\epsilon : \mathbb{Z} \rightarrow [0, 1]$ that determines the probability of sampling the true observation over the previous prediction. When ϵ is 1.0, the algorithm behaves akin to teacher forcing [Williams and Zipser, 1989] (with the same action-conditional modification); whereas, as it approaches 0.0 it becomes fully auto-regressive.

9.2.2 World Model Implementation

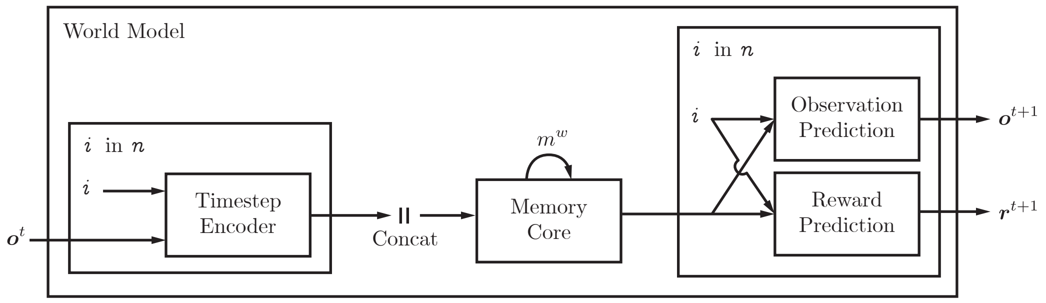


Figure 9.1: **World Model Architecture.**

The high-level architecture of the world model is illustrated in Figure 9.1. The world model is composed of several modules that are quite similar to the policy:

- *Timestep Encoder*: Processes all of the current observation's information into a single embedding vector. The timestep includes all new observational data that the agent gains at the current point in time. Different from the agent's timestep encoder, this encoder also receives the ID that corresponds with the timestep.

- *Memory Core*: The component of the agent that maintains and update's the agent's memory. Different from the agent's timestep encoder, this memory core receives the representation of each player's timestep concatenated.

- *Observation Prediction (Head)*: Predicts the successor observation for each player. As all games considered in this work are gridworld games, the predicted observation is a classification task for each future grid cell (that are within the respective player's observation window).

- *Reward Prediction (Head)*: Predicts the reward received for each player. Rewards are treated as categorical values.

Note, that the timestep encoder, observation prediction head, and reward prediction head each use the same parameters across each player. Similar to the agent, all components are simultaneously trained and their joint parameters are referred to as $\theta^w \in \Theta^w$. Both observation and reward losses are optimized with a cross entropy objective, and averaged across players. The total world model loss is as follows:

$$\mathcal{L}_w = \lambda_{\text{observation}} \cdot \mathcal{L}_{\text{observation}} + \lambda_{\text{reward}} \cdot \mathcal{L}_{\text{reward}}.$$

The implementation of each component is as follows:

- *Timestep Encoder*: The same as the agent's timestep encoder, but the player's ID is also provided alongside the action into the second neural network.
- *Memory Core*: Identical to the agent.
- *Observation Prediction (Head)*: The observation prediction is based on the memory core's output and a one-hot ID of the predicted player's ID. These inputs are concatenated and fed into an transposed version of the timestep encoder.
- *Reward Prediction (Head)*: A linear layer of size one. For Harvest: Categorical this output is handled as a discrete prediction; whereas, it is continuous for the other games.

The world model is implemented using JAX [Bradbury et al., 2018] with Haiku modules [Hennigan et al., 2020]. The dataset used to train the world model is an Reverb replay buffer [Cassirer et al., 2021].

A world model is trained for $1{,}250{,}000$ updates. Each example in the mini-batch is a sequence of 20 transitions, where the first 5 timesteps are used to burn-in the memory. Burn-in does not occur for examples where the first 5 transitions are at the beginning of the episode. Moreover, sequences are added into the replay buffer at a period of 14 so that all timesteps show up as prediction targets.

The world model is trained using action-conditioned scheduled sampling, Algorithm 5. The schedule ϵ follows the following schedule:

$$\epsilon(t) = \begin{cases} 1.0 & t < 250000 \\ \frac{4}{3} - \frac{t}{750000} & 250000 \leq t \leq 1000000 \\ 0.0 & t > 1000000. \end{cases}$$

This schedule starts out training as a variation of teacher forcing [Williams and Zipser, 1989], and slowly transitions to fully auto-regressive. Additional hyperparameters are specified in Table 9.1.

Table 9.1: **World model hyperparameters per game.** Adam was introduced by Kingma and Ba [2015].

Hyperparameter	Harvest: Categorical	Harvest: RGB	Running with Scissors
$\lambda_{\text{observation}}$	1.0	1.0	1.0
λ_{reward}	10.0	0.01	0.01
Optimizer	Adam	Adam	Adam
Learning Rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Max Grad Norm	10.0	10.0	10.0
Batch Size	32	24	24

9.3 Findings

Results. Figure 9.2 presents each world model's per-profile accuracy, as well as its average over all profiles. Inclusion of the random policy corresponds to decreases in observation prediction accuracy:

⊡ $0.75 \pm 0.02 \rightarrow$ ◰ 0.58 ± 0.05,
⊡ $0.80 \pm 0.02 \rightarrow$ ⁘ 0.62 ± 0.05,
▣ $0.83 \pm 0.02 \rightarrow$ ◼ 0.68 ± 0.04.

Figure 9.3 contains the world model's per-profile recall. Inclusion of the random policy corresponds to increases in reward 1 recall:

⊡ $0.25 \pm 0.07 \rightarrow$ ◰ 0.37 ± 0.11,
⊡ $0.25 \pm 0.07 \rightarrow$ ⁘ 0.36 ± 0.11,
▣ $0.26 \pm 0.07 \rightarrow$ ◼ 0.37 ± 0.11.

Figure 9.4 graphically depicts these trends.

We verify the diversity in our set of policies by measuring their action agreement. The method that we do this is follows. We begin by simulating every pair of policies for 30 episodes each. We then collect the action that each policy would select for all observations across all episodes. The similarity between a pair of policies is the fraction of the same actions they would have taken. Table 9.2 contains the similarity results across all episodes. Table 9.3 contains the similarity results excluding the episodes played with a random policy. This secondary comparison highlights the similarity of the policies biased towards more strategically salient episodes.

We further compare our world models by their cross-entropy loss over the full dataset. Figure 9.5 contains the full per-profile comparison, and Figure 9.6 contains an aggregate comparison.

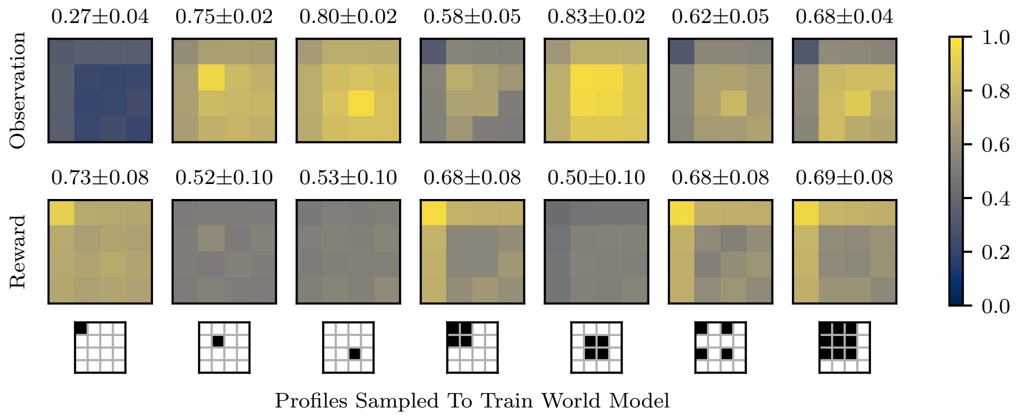


Figure 9.2: **World model accuracy across strategy profiles.** Each heatmap portrays a world model's accuracy over 16 strategy profiles. The meta x-axis corresponds to the profiles used to train the world model (as black cells). Above each heatmap is the model's average accuracy.

Table 9.2: **Policy similarity measured by action agreement.**

Policy ID	0	1	2	3
0	1.0000	0.1262	0.1277	0.1279
1		1.0000	0.8868	0.8269
2			1.0000	0.8961
3				1.0000

Table 9.3: **Policy similarity measured by action agreement without random observations generated from random policies.**

Policy ID	0	1	2	3
0	1.0000	0.1265	0.1272	0.1283
1		1.0000	0.8542	0.7671
2			1.0000	0.8608
3				1.0000

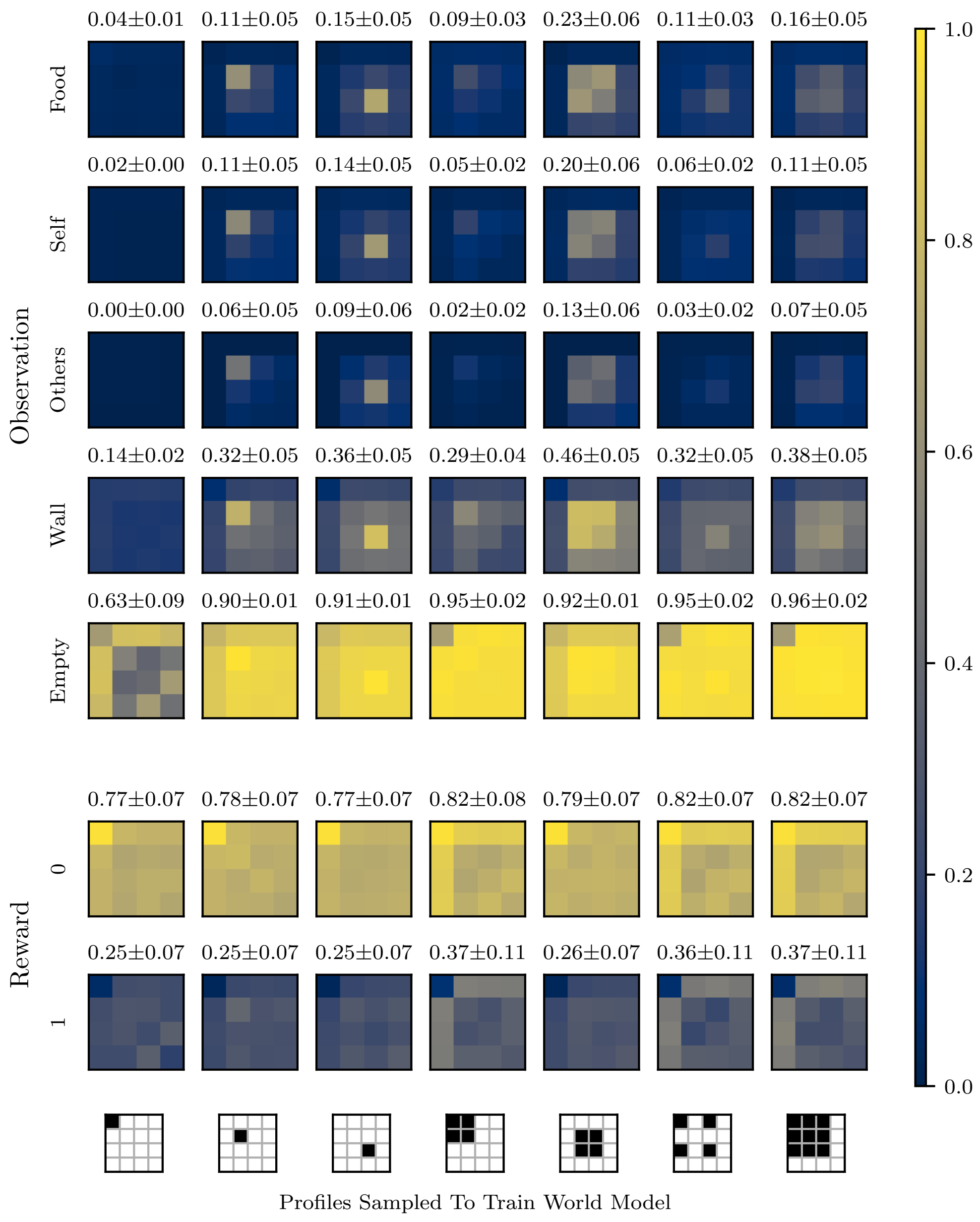


Figure 9.3: **World model recall across strategy profiles.** Each heatmap portrays a world model's recall over 16 strategy profiles. The meta x-axis corresponds to the profiles used to train the world model (as black cells). Above each heatmap is the model's average recall.

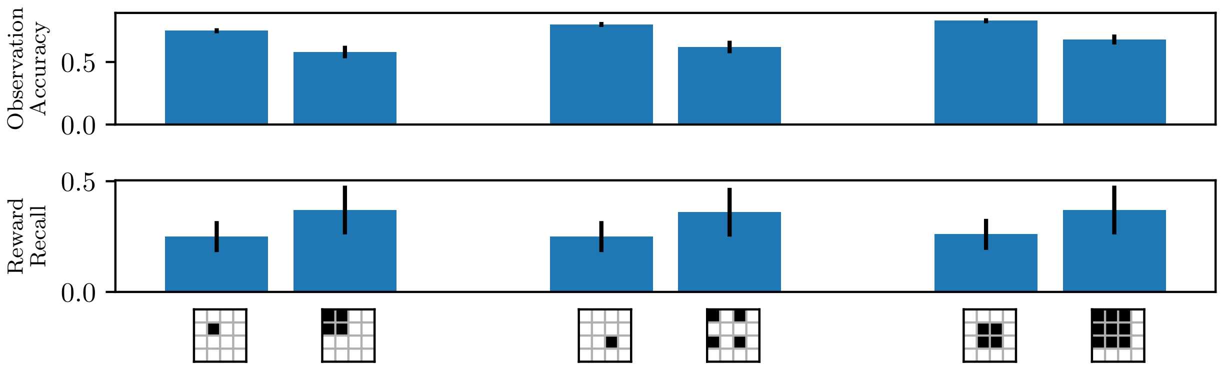


Figure 9.4: **Impact of inclusion of random policy on world model performance.**

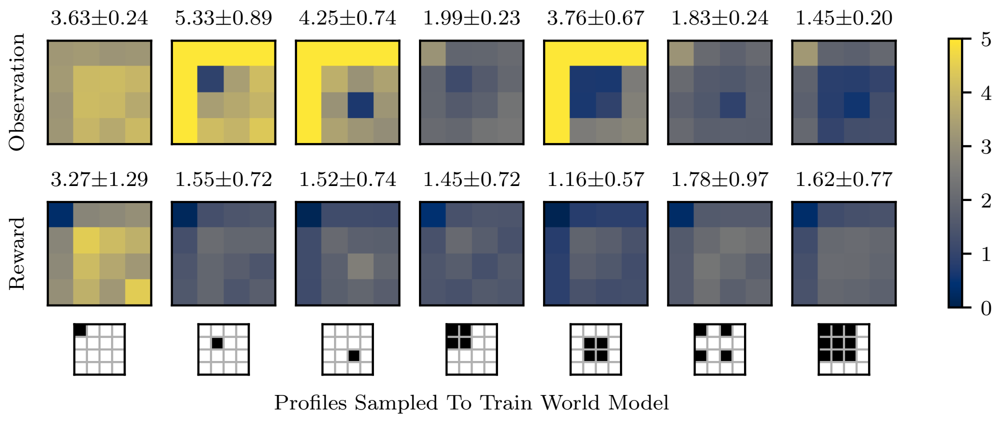


Figure 9.5: **Per-profile comparison of world models by their cross-entropy loss.**

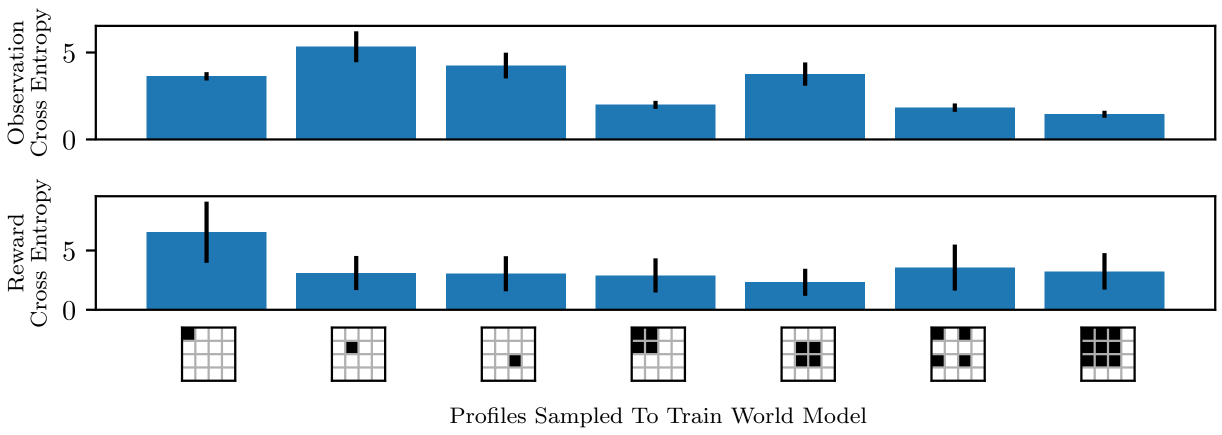


Figure 9.6: **Aggregate comparison of world models by their cross-entropy loss.**

Discussion. The PSRO policies offer the most strategically salient view of the game's dynamics. Consequently, the world model ▣ trained with these policies yields the highest observation accuracy. However, this world model performs poorly on reward accuracy, scoring only 0.50 ± 0.10. In comparison, the model trained on the random policy ▦ scores 0.73 ± 0.08. This seemingly counterintuitive result can be attributed to a significant class imbalance in rewards. ▦ predicts only the most common class, no reward, which gives the illusion of higher performance. In contrast, the remaining world models attempt to predict rewarding states, which reduces their overall accuracy. Therefore, we should compare the world models based on their ability to recall rewards. When we examine ▣ again, we find that it also struggles to recall rewards, scoring only 0.26 ± 0.07. However, when the random policy is included in the training data (◼), the recall improves to 0.37 ± 0.11. This improvement is also due to the same class imbalance. The PSRO policies are highly competitive, tending to over-harvest. This limits the proportion of rewarding experiences. Including the random policy enhances the diversity of rewards in this instance, as its coplayer can demonstrate successful harvesting. When we compare the world models by their evaluation loss (Figure 9.5), ◼ obtains the lowest observation loss of 1.45 ± 0.20, whereas, ▣ obtains the lowest reward loss of 1.16 ± 0.57. Given the importance of accurately predicting both observations and rewards for effective planning, ◼ appears to be the most promising option. However, the strong performance of ▣ suggests future work on algorithms that can benefit solely from observation predictions. Overall, these results support the claim that strategic diversity enhances the training of world models.

CHAPTER 10

Empirical Games & Planning

In this dissertation I will investigate transferring a world model through a process called *planning*. Planning is any procedure that takes a world model and produces or improves a policy. In the context of games, planning can optionally take into account the existence of coplayers. This consideration can reduce experiential variance caused by unobserved confounders (i.e., the coplayers). However, coplayer modeling errors may introduce further errors in the planning procedure [He et al., 2016].

Planning alongside empirical-game construction allows us to side-step this issue as we have direct access to the policies of all players during training. This allows us to circumvent the challenge of building accurate coplayer models. Instead, the policies of coplayers can be directly queried and used alongside a world model, leading to more accurate planning. In this section, we empirically demonstrate the effectiveness of two methods that decrease the cost of response calculation by integrating planning with a world model and other agent policies.

10.1 Experimental Details

In this section, I discuss the agents within this part of the dissertation. I first detail the implementation of the agents and then their training.

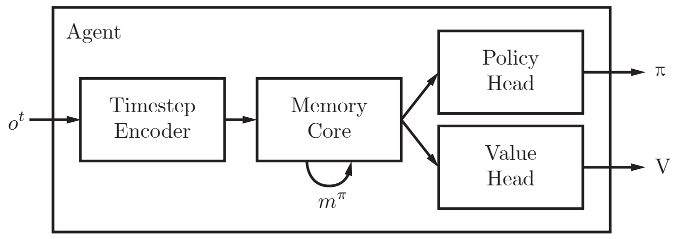


Figure 10.1: **Agent Architecture.**

All agents follow the general architecture depicted in Figure 10.1. This consists of four modules:

- *Timestep Encoder*: Processes all of the current observation's information into a single embedding vector. The timestep includes the new observation and the policy's previous action.
- *Memory Core*: The component of the agent that maintains and update's the agent's memory.
- *Policy Head*: Computes the agent's policy.
- *Value Head*: Computes the agent's state value function.

All of the components are simultaneously trained and their joint parameters are $\theta^{\pi} \in \Theta^{\pi}$. The policies are trained using the IMPALA algorithm [Espeholt et al., 2018]. For the IMPALA loss, the coefficients for each component loss are:

$$\mathcal{L}_{\text{IMPALA}} = \lambda_{\pi} \cdot \mathcal{L}_{\pi} + \lambda_{\text{V}} \cdot \mathcal{L}_{\text{V}} + \lambda_{\text{entropy}} \cdot \mathcal{L}_{\text{entropy}},$$

with a discount factor of 0.99. Similar to the world models, the policies are implemented using JAX [Bradbury et al., 2018] with Haiku modules [Hennigan et al., 2020]. Reverb is used as the replay buffer implementation [Cassirer et al., 2021].

The training details for each specific response calculation are itemized below.

Baseline Parameters The learning rate begins is linearly decayed over $10{,}000$ updates. Each update is computed from a mini-batch of 128 examples that are generated from 8 arenas[1] Policy parameters are synchronized at the beginning of each episode. Each example in the mini-batch is a sequence of 20 transitions. Moreover, sequences are stored in a replay buffer with a period of 19, to ensure that the action played at the end of a sequence is trained. Sequences are stored in a replay buffer with a max capacity of $1{,}000{,}000$, and are evicted once sampled. Additional hyperparameters are specified in Table 10.1.

Harvest: Categorical module implementations:

- *Timestep Encoder*: The encoder processes two timestep components: the current observation and the previous action the policy took. First the observation is passed through a two-layer fully connected neural network with hidden sizes of $[256, 256]$. The representation of the observation is then concatenated with the previous action (represented as a one-hot vector), and passed together through a second neural network with sizes $[256, 256]$. All of the layers

[1]The term *arena* is used to refer to an experience generation process. This is more commonly referred to as an "actor"; however, this terminology may be confounding with language in RL, Dyna, or multiagent learning.

Table 10.1: **Baseline agent's hyperparameters per game.** See Kingma and Ba [2015] for Adam, and Hinton [2018] for RMSProp.

Hyperparameter	Harvest: Categorical	Harvest: RGB	Running with Scissors
Optimizer	Adam	RMSProp	RMSProp
λ_π	1.0	1.0	1.0
λ_V	0.2	0.5	0.2
λ_entropy	0.04	0.01	0.003
Learning Rate Start	6×10^{-6}	6×10^{-4}	1×10^{-4}
Learning Rate Stop	6×10^{-9}	6×10^{-9}	1×10^{-4}
Max Grad Norm	10.0	1.0	0.1
Batch Size	128	128	128

have ReLU [Fukushima, 1975] activations including the final layers of both networks. The final representation is the output of the timestep encoder.

- *Memory Core*: A single-layer LSTM [Hochreiter and Schmidhuber, 1997] with $256\,\text{units}$.
- *Policy Head*: A single linear layer of size 8.
- *Value Head*: A single linear layer of size 1.

Harvest: RGB and Running with Scissors module implementations:

- *Timestep Encoder*: The encoder processes two timestep components: the current observation and the previous action the policy took. The observation is first process by a two-layer convolutional neural network with ReLU activations [Fukushima, 1975]. The first layer has 16 channels, a kernel with shape $[8, 8]$, and a stride of $[8, 8]$. The second layer has 32 channels, a kernel shape of $[4, 4]$, and a stride of $[1, 1]$. The output of this layer is then flattened and concatenated with a one-hot encoding of the policy's previous action. The resulting embedding is then passed through a two-layer fully connected neural network with hidden sizes of $[128, 128]$, and ReLU activations.
- *Memory Core*: A single-layer LSTM [Hochreiter and Schmidhuber, 1997] with $128\,\text{units}$.
- *Policy Head*: A single linear layer of size 8.
- *Value Head*: A single linear layer of size 1.

Planning Parameters The planners have the same hyperparameters as the baseline method, but with the addition of planning-specific settings. For all planners, an additional 4 arenas are used to generate planned experiences (for background planning). The additional settings for each version of planning are as follows:

- *Warm-Start Background Planning*: An additional $10{,}000$ updates are performed on exclusively planned experiences before play in the real game occurs.
- *Concurrent Background Planning*: Each mini-batch sampled after warm-starting contains $25\,\%$ planned experiences, and $75\,\%$ real experiences.
- *Decision-Time Planning*: In the training arenas (those that have the real game, and are not used for evaluation), the agent selects actions with a beam-search of width 3 and depth 1.

Background planning also requires defining a *search control* procedure [Sutton, 1990, 1991, Sutton and Barto, 2018]. Search control defines how the agent prioritizes selecting starting states and actions for background planning. A related concept is that of prioritized sweeping, which defines how an agent prioritizes using sampled data for training [Moore and Atkeson, 1993]. The notable difference being search control alters the data generation process; whereas, prioritized sweeping alters how that data is prioritized for learning. This work considers the simplest search-control method: maintain a buffer of the initial states and uniformly sample.

10.2 Background Planning

The first type of planning that is investigated is *background planning*, popularized by the Dyna architecture [Sutton, 1990]. In background planning, agents interact with the world model to produce *planned experiences*[2]. The planned experiences are then used by a model-free reinforcement learning algorithm as if they were *real experiences* (experiences generated from the real game). Background planning enables learners to generate experiences of states they are not currently in.

Experiment. To assess whether planned experiences are effective for training a policy in the actual game, we compute two response policies. The first response policy, serving as our baseline, learns exclusively from real experiences. The second response policy, referred to as the planner, is trained using a two-step procedure. Initially, the planner is exclusively trained on planned experiences. After $10{,}000$ updates, it then transitions to learning solely from real experiences. The planner employs the ▛ world model from Chapter 9, and the opponent plays the previously held-out policy. In this and subsequent experiments, the cost of methods is measured by the number of experiences they require with the actual game. This is because, experience collection is often the bottleneck when applying RL-based methods [Obando-Ceron and Castro, 2021, Hester and

[2]Other names include “imaginary”, “simulated”, or “hallucinated” experiences.

Stone, 2012]. Throughout the remainder of this work, each experience represents a trajectory of 20 transitions, facilitating the training of recurrent policies.

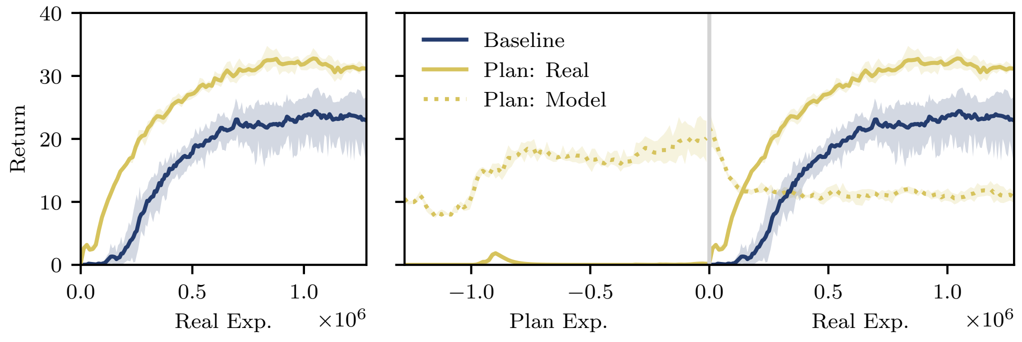


Figure 10.2: **Effects of background planning on response learning using ▛.** Left: Return curves measured by the number of real experiences used. Right: Return curves measured by usage of both real and planned experiences. The planner's return is measured against the real game and the world model. (5 seeds, with 95 % bootstrapped CI).

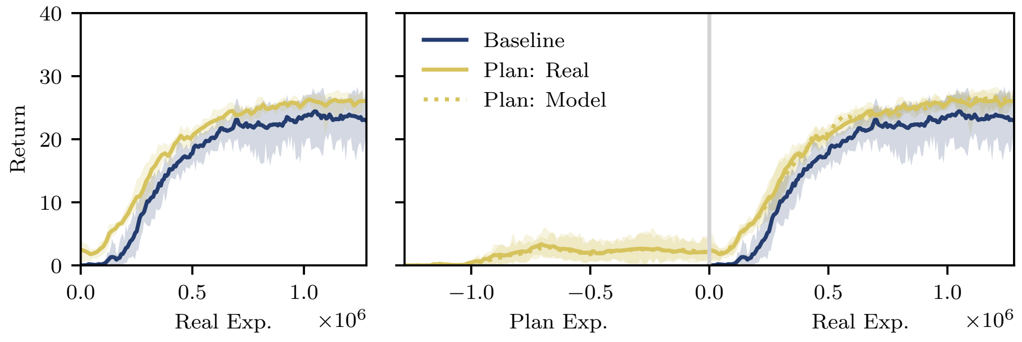


Figure 10.3: **Effects of background planning on response learning using ▦.** Left: Return curves measured by the number of real experiences used. Right: Return curves measured by usage of both real and planned experiences. The planner's return is measured against the real game and the world model. (5 seeds, with 95 % bootstrapped CI).

Results. Figure 10.2 presents the results of the background planning experiment. The methods are compared based on their final return, utilizing an equivalent amount of real experiences. The baseline yields a return of 23.00 ± 4.01, whereas the planner yields a return of 31.17 ± 0.25.

Discussion. In this experiment, the planner converges to a stronger policy, and makes earlier gains in performance than the baseline. Despite this, there is a significant gap in the planner's learning curves, which are reported with respect to both the world model and real game. This gap arises due to accumulated model-prediction errors, causing the trajectories to deviate from the true state space. Thereby, this gap represents an existing limitation in the world model methodology that we employ. Advances in world modeling methods may further bridge this gap, and improve the benefits that can be gained by planning. Nevertheless, the planner effectively learns to interact with the world model during planning, and this behavior shows positive transfer into the real game, as evidenced by the planner's rapid learning. The exact magnitude of benefit will vary across coplayers' policies, games, and world models. In Figure 10.3, we repeat the same experiment with the poorly performing ▦ world model, and observe a marginal benefit (26.05 ± 1.32). The key take-away is that background planning tends to lead towards learning benefits, and not generally hamper learning.

10.3 Decision-Time Planning

The second main way that a world model is used is to inform action selection at *decision time [planning] (DT)*. In this case, the agent evaluates the quality of actions by comparing the value of the model's predicted successor state for all candidate actions. Action evaluation can also occur recursively, allowing the agent to consider successor states further into the future. Overall, this process should enable the learner to select better actions earlier in training, thereby reducing the amount of experiences needed to compute a response. A potential flaw with decision-time planning is that the agent's learned value function may not be well-defined on model-predicted successor states [Talvitie, 2014]. To remedy this issue, the value function should also be trained on model-predicted states.

Experiment. To evaluate the impact the decision-time planning, we perform an experiment similar to the background planning experiment (Section 10.2). However, in this experiment, we evaluate the quality of four types of decision-time planners that perform one-step three-action search. The planners differ in the their ablations of background planning types: (1) *warm-start background planning (BG: W)* learning from planned experiences before any real experiences, and (2) *concurrent background planning (BG: C)* where after BG: W, learning proceeds simultaneously on both planned and real experiences. The intuition behind BG: C is that the agent can complement its learning process by incorporating planned experiences that align with its current behavior, offsetting the reliance on costly real experiences.

Results. The results for this experiment are shown in Figure 10.4. The baseline policy receives a final return of 23.00 ± 4.01. The planners that do not include BG: W, perform worse, with final returns of 9.98 ± 7.60 (DT) and 12.42 ± 3.97 (DT & BG: C). The planners that perform BG: W

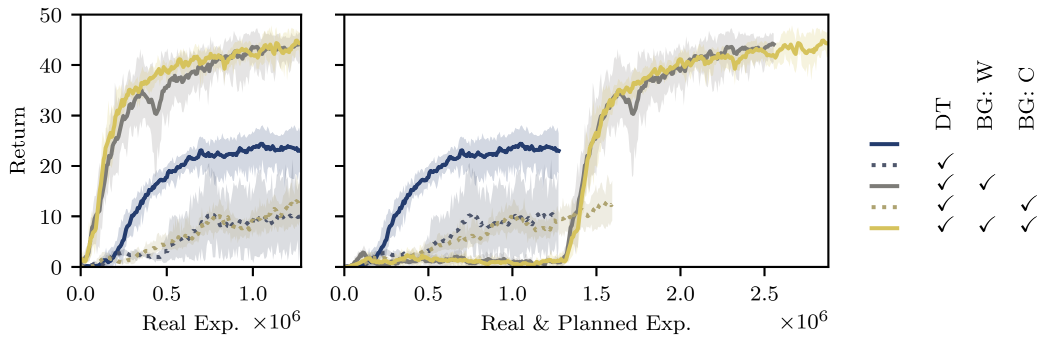


Figure 10.4: **Effects of decision-time planning on response learning using ▚.** Four planners using decision-time planning (DT) are shown in combinations with warm-start background planning (BG: W) and concurrent background planning (BG: C). (5 seeds, with $95\,\%$ bootstrapped CI).

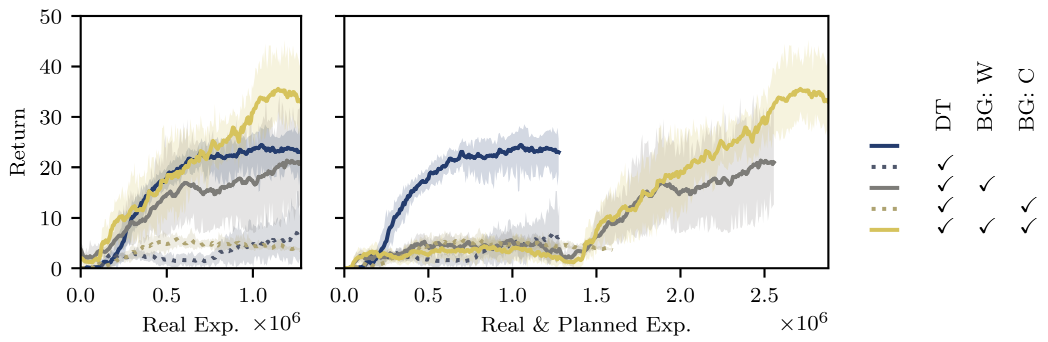


Figure 10.5: **Effects of decision-time planning on response learning using ▦.** Four planners using decision-time planning (DT) are shown in combinations with warm-start background planning (BG: W) and concurrent background planning (BG: C). (5 seeds, with $95\,\%$ bootstrapped CI).

outperform the baseline, with final returns of 44.11 ± 2.81 (DT & BG: W) and 44.31 ± 2.56 (DT, BG: W, & BG: C).

Discussion. Our results suggest that the addition of BG: W provides sizable benefits:

9.98 ± 7.60 DT $\rightarrow$ 44.11 ± 2.81 DT & BG:W, and

12.42 ± 3.97 DT & BG: C $\rightarrow$ 44.31 ± 2.56 DT, BG: W, & BG: C.

We postulate that this is because it informs the policy's value function on model-predictive states early into training. This allows that the learner is able to more effectively search earlier into training. BG: C appears to offer minor stability and variance improvements throughout the training procedure; however, it does not have a measurable difference in final performance. This result

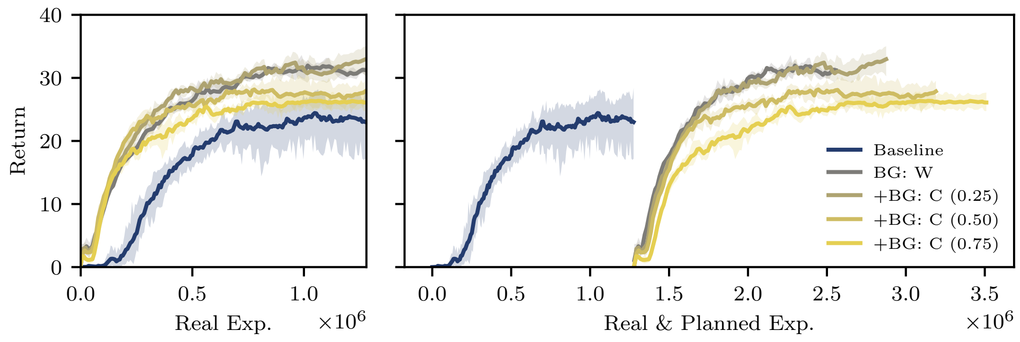


Figure 10.6: **Effects of concurrent background planning on response computation using world model ■.** Methods labelled "+BG: C (X)" perform both BG: W and BG: C, where X denotes the proportion of planned experience within in batch of data. (5 seeds with 95 % bootstrapped CI).

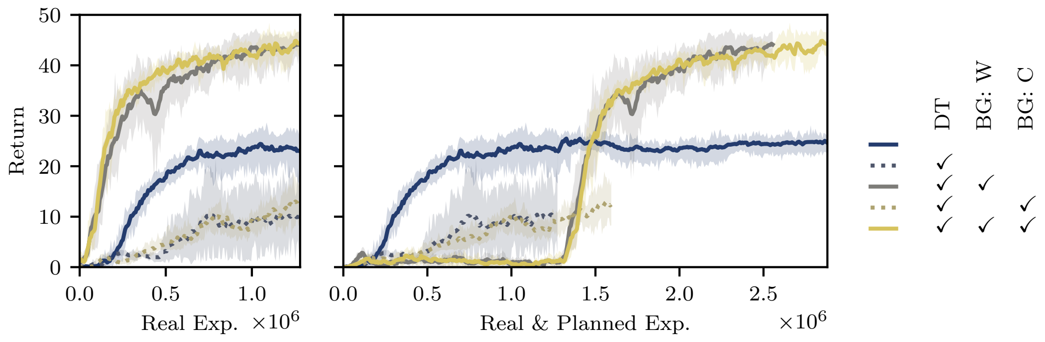


Figure 10.7: **Decision-time planning using ■ compared against a baseline trained longer.** (5 seeds, with 95 % bootstrapped CI.)

suggests using planning methods in combination to reap their respective advantages.

However, we caution against focusing on the magnitude of improvement found within this experiment. As the margin of benefit depends on many factors including the world model accuracy, the opponent policy, and the game. To exemplify, similar to the background planning section, we repeat the same experiment with the poorly performing ▦ world model. The results of this ancillary experiment are in Figure 10.5. The trend of BG: W providing benefits was reinforced:

$$\begin{array}{lcl} 6.29 \pm 5.12 \text{ DT} & \rightarrow & 20.98 \pm 9.76 \text{ DT \& BG: W, and} \\ 3.64 \pm 0.26 \text{ DT \& BG: C} & \rightarrow & 33.07 \pm 7.67 \text{ DT, BG: W, \& BG: C.} \end{array}$$

However, the addition of BG: C now measurably improved performance

$$20.98 \pm 9.76 \text{ DT \& BG: W} \rightarrow 33.07 \pm 7.67 \text{ DT, BG: W, \& BG: C.}$$

An additional ablation was performed on BG: C to understand its role alongside BG: W. These results are shown in Figure 10.6 where we vary the intensity of BG: C by training planners that train off larger proportions of planned experience. Without DT, we observed no measurable benefit of including BG: C. As the proportion of planned experience increases in BG: C this corresponded to a decrease in the planner's performance. We speculate that this is because the planner better fits its policy to interact with the world model instead of the real game. This result suggests that BG: C be used to only supplement a small fraction of the real experiences used during training.

Finally, we completed an ancillary experiment to determine if planning allowed the learner to escape a locally optimal policy. To measure this we simply continued training the baseline on a significantly larger data budget to measure if its performance would improve, and even match that of the planner. In Figure 10.7 we plot our results, and found that DT planning helped learner a stronger policy than a learner without planning.

The main outcome of these experiments is the observation that multi-faceted planning is unlikely to harm a response calculation, and has a potentially large benefit when applied effectively. These results support the claim that world models offer the potential to improve response calculation through decision-time planning.

10.4 Separate Models

Until now, we have implicitly assumed the need for distinct models. However, if a single model could serve both functions, co-learning two separate models would not be needed.

Empirical games, in general, cannot replace a world model as they entirely abstract away any concept of game dynamics. This is not without any exceptions. If the original game is one-shot and stateless (i.e., an episode is played through a single action), then a normal-form empirical game is exactly a world model.

Conversely, world models have the potential to substitute for the payoff estimations in empirical games through simulated rollouts. Note, that rolling out a trajectory with a world model is an auto-regressive prediction that tends to result in compounding errors [Talvitie, 2014, Holland et al., 2018]. Despite this, it is plausible that a world model can substitute as a high-fidelity empirical game.

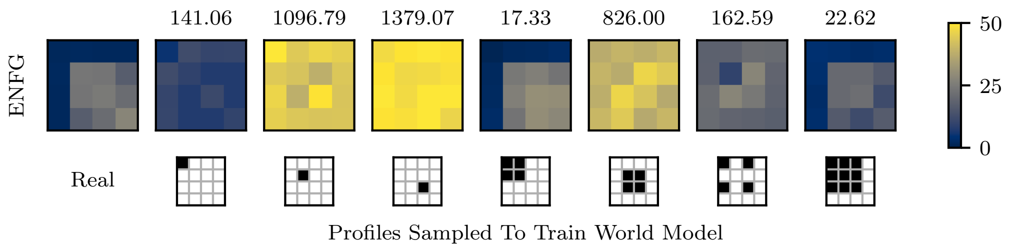


Figure 10.8: **Empirical normal-form games (ENFG) estimated by world model rollouts.** The title of each plot is its L2 distance with the real ENFG.

Figure 10.8 compares an empirical game estimated from real game payouts empirical games estimated with payouts predicted by a world model. In this experiment, the world models are the same that were used in Chapter 9. In general, the empirical games estimated by world models have large errors (L2 >100), with several having exceptionally large errors (L2 >1000). These result suggest that this direction may be possible with future algorithmic improvements; however, currently, the prediction errors are too large to substitute empirical games with world models. Especially in games with long time horizons.

Having defined the models and established the need for their separate instantiations, we can proceed to evaluate the claims of beneficial co-learning. Our first experiment shows that the strategic diversity embodied in an empirical game yields diverse game dynamics, resulting in the training of a more performant world model. The second set of experiments demonstrates that a world model can help reduce the computational cost of policy construction in an empirical game.

CHAPTER 11

Game Solving

In this section we introduce Dyna-PSRO, *Dyna*-Policy-Space Response Oracles, an approximate game-solving algorithm that builds on the PSRO [Lanctot et al., 2017] framework. Dyna-PSRO employs co-learning to combine the benefits of world models and empirical games.

Dyna-PSRO is defined by two significant alterations to the original PSRO algorithm. First, it trains a world model in parallel with all the typical PSRO routines (i.e., game reasoning and response calculation). We collect training data for the world model from both the episodes used to estimate the empirical game's payoffs, and the episodes that are generated during response learning and evaluation. This approach ensures that the world model is informed by a diversity of data from a salient set of strategy profiles. By reusing data from empirical game development, training the world model incurs no additional cost for data collection.

The second modification introduced by Dyna-PSRO pertains to the way response policies are learned. Dyna-PSRO adopts a Dyna-based reinforcement learner [Sutton, 1990, 1991, Sutton et al., 2012] that integrates simultaneous planning, learning, and acting. Consequently, the learner concurrently processes experiences generated from decision-time planning, background planning, and direct game interaction. These experiences, regardless of their origin, are then learned from using the IMPALA [Espeholt et al., 2018] update rule. For all accounts of planning, the learner uses the single world model that is trained within Dyna-PSRO. This allows game knowledge accrued from previous response calculations to be transferred and used to reduce the cost of the current and future response calculations.

11.1 Dyna-PSRO

The Dyna-PSRO builds upon PSRO (Algorithm 7) by including the co-learning of a world model. The high-level pseudocode of Dyna-PSRO is provided in Algorithm 9 an a high-level application architecture diagram is depicted in Figure 11.1. There are three main co-routines of Dyna-PSRO: response computation, world-model learning, and empirical-game simulation. The details of each routine are first provided; then, how the routines interact with each other is explained. We implement Dyna-PSRO using LaunchPad [Yang et al., 2021a] to distribute the coroutines in a design that is inspired by the ACME library [Hoffman et al., 2020].

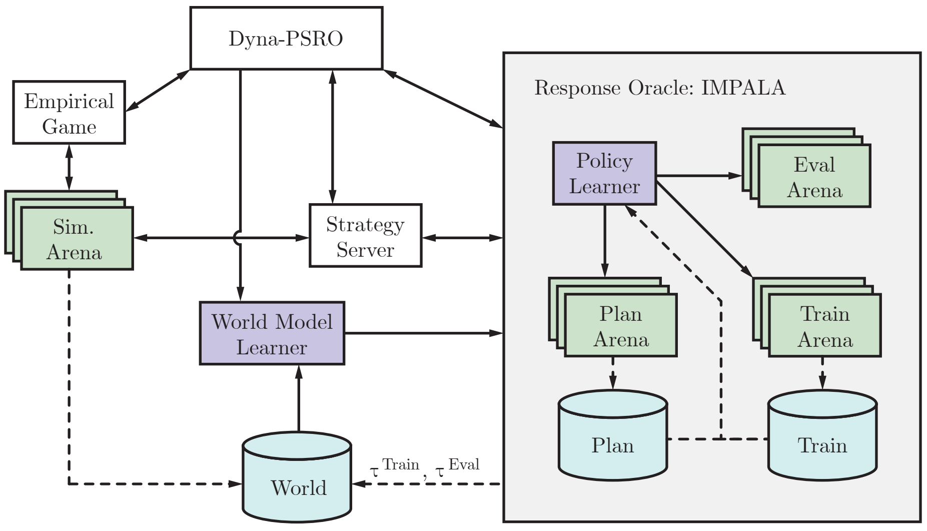


Figure 11.1: **Overview of the major Dyna-PSRO processes.**

11.1.1 Empirical Game

The empirical game routine is responsible for maintaining the empirical game, including simulating new payoffs and game reasoning. New profiles are sent to *simulation (sim.) arenas* for payoff estimation in parallel. Once all profiles are estimated, the game is solved, and the solution is based to the main Dyna-PSRO process. In the experiments in this work, the chosen solution is Nash Equilibrium, and it is solved through the linear complementarity [Eaves, 1971] algorithm that is implemented by Gambit [McKelvey et al., 2016].

11.1.2 World Model

The world model routine is responsible for training the world model and serving its parameters. This routine's pseudocode is provided in Algorithm 6, and follows mostly the same method details as the strategic diversity experiment. The difference is that instead of there being a precomputed fixed dataset, the world model is now trained over a dynamic dataset. The dataset is represented by a replay buffer that is populated from:

1. trajectories from the simulation arena used for expanding the empirical game, and

2. trajectories from the training and evaluation arenas from the response calculation.

Notably, all of this data must be generated in the standard PSRO procedure, so it collected with no additional cost. The world-model learner samples and evicts data randomly from this buffer.

Algorithm 6: World Model Learner

```
Input: World model $w$ and data buffer $\mathcal{B}^w$
Input: $n$ no. of updates (default: $\infty$).
for $i \in [[n]]$ do
    Train $w$ over $\tau \sim \mathcal{B}^w$;
Output: $w$
```

11.1.3 Response Oracle

The response oracle uses the IMPALA [Espeholt et al., 2018] algorithm to compute an approximate best-response to the opponent's strategy according the the current empirical game. IMPALA uses several processes that generate experiences for the agent to train on. These process are referred to in this work as arenas. The *train arenas* generate real experiences, and the *plan arenas* generate planned experiences. If the learner is using decision-time planning they will only use it in the train arenas. A third set of arenas called *eval arenas* periodically evaluate the performance of the greedy policy and record additional metrics. The arenas attempt to synchronize all parameters at the start of each episode.

The policy learner runs for a fixed number of updates, querying the datastores for experiences to learn from. The specifics of how each policy learns is described in Section 10.1.

Algorithm 7: PSRO with Subroutine

```
Input: Initial strategy sets for all players $\mathbf{\Pi}^0$
Simulate utilities $\hat{U}^{\mathbf{\Pi}^0}$ for each joint $\boldsymbol{\pi}^0 \in \mathbf{\Pi}^0$;
Initialize solution $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$;
while epoch $e$ in $\{1, 2, \dots\}$ do
    for player $i \in [[n]]$ do
        // Algorithm 8.
        $\pi_i^e, \_ = \text{response\_oracle}(\sigma_{-i}^{*,e-1})$;
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$;
    Simulate missing entries in $\hat{U}^{\Pi^e}$ from $\mathbf{\Pi}^e$;
    Compute a solution $\sigma^{*,e}$ from $\hat{\Gamma}^e$;
Output: Current solution $\sigma_i^{*,e}$ for player $i$
```

Algorithm 8: Response Oracle

```
Input: Coplayer strategy profile $\sigma_{-i}$
Input: Num updates $k$
$\pi_i \leftarrow \theta^\pi$;
$\mathcal{B} \leftarrow \{\}$ ; // Replay Buffer.
for many async episodes do
    $\pi_{-i} \sim \sigma_{-i}$;
    $\mathcal{B} = \mathcal{B} \cup \{\tau \sim (\pi_i, \pi_{-i})\}$;
for $i \in [[k]]$ do
    Train $\pi_i$ over $\tau \sim \mathcal{B}$;
Output: $\pi_i, \mathcal{B}$
```

Algorithm 9: Dyna-PSRO

```
Input: Initial strategy sets for all players $\mathbf{\Pi}^0$
Input: No. of world model head-start updates $n_w$
Input: Epoch to begin planning $e^{\text{plan}}$
Simulate utilities $\hat{U}^{\mathbf{\Pi}^0}$ for each joint $\boldsymbol{\pi}^0 \in \mathbf{\Pi}^0$;
Initialize solution $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$;
$w \leftarrow \theta^w$;
$\mathcal{B}^w \leftarrow \{\}$ ;                    // World Model's Replay Buffer.
;
while epoch e in $\{1, 2, \dots\}$ do
    for player $i \in [[n]]$ do
        if $e > e^{\text{plan}}$ then
            $\pi_i^e, \tau = \text{async}(\text{planner\_oracle}(\sigma_{-i}^{*,e-1}, w))$ ;        // Algorithm 10.
        else
            $\pi_i^e, \tau = \text{async}(\text{response\_oracle}(\sigma_{-i}^{*,e-1}))$ ;        // Algorithm 8.
        $\mathcal{B}^w = \mathcal{B}^w \cup \{\tau\}$;
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$;
    Wait on all futures $\boldsymbol{\pi}^e, \tau$;
    ;
    Simulate missing entries in $\hat{U}^{\mathbf{\Pi}^e}$ from $\mathbf{\Pi}^e$;
    Add $\tau$ from simulation to $\mathcal{B}^w$;
    Compute a solution $\boldsymbol{\sigma}^{*,e}$ from $\hat{\Gamma}^e$;
    ;
    if e == 1 then
        $w = \text{world\_model\_learner}(w, n_w)$ ;                 // Algorithm 6.
        $w = \text{async}(\text{world\_model\_learner}(w))$ ;    // Parameters periodically
         sync.
Output: Current solution $\sigma_i^{*,e}$ for player $i$
```

```
Algorithm 10: Planner Oracle
  Input: Coplayer strategy profile $\sigma_{-i}$
  Input: World model $w$, real game dynamics $p$
  Input: Warm-start background planning updates $n^{\text{BG:WS}}$
  Input: Training updates $n$
  Input: Concurrent background planning fraction $f^{\text{BG:C}}$
  $\pi_i \leftarrow \theta^{\pi}$;
  $\mathcal{B}^{\text{plan}} \leftarrow \{\}$ ;          // Replay Buffer with planned experience.
  $\mathcal{B}^{\text{train}} \leftarrow \{\}$ ;            // Replay Buffer with real experience.
  ;
  // Asynchronously generate data on arenas.;
  for many async episodes do
      $\pi_{-i} \sim \sigma_{-i}$;
      $\mathcal{B}^{\text{plan}} = \mathcal{B}^{\text{plan}} \cup \{\tau \sim (\pi_i, \pi_{-i}, w)\}$;
  for many async episodes do
      $\pi_{-i} \sim \sigma_{-i}$;
      $\mathcal{B}^{\text{train}} = \mathcal{B}^{\text{train}} \cup \{\tau \sim (\pi_i, \pi_{-i}, p)\}$;
  ;
  // Train the response policy.;
  for $i \in [[n^{\text{BG:WS}}]]$ do
      Train $\pi_i$ over $\tau \sim \mathcal{B}^{\text{plan}}$;
  for $i \in [[n]]$ do
      Train $\pi_i$ over $\tau \sim \left\{(1.0 - f^{\text{BG:C}}) \cdot \mathcal{B}^{\text{train}}\right\} \cup \left\{f^{\text{BG:C}} \cdot \mathcal{B}^{\text{plan}}\right\}$;
  Output: $\pi_i, \mathcal{B}$
```

11.1.4 Runtime Procedure

A sketch of the respective processes runtime is shown in Figure 11.2 As in PSRO, the main empirical-game building loop iterates between response computation and empirical-game simulation.

The runtime is defined by a parameter specifying on which PSRO epoch to begin planning. Before that epoch, the response oracles do not use planning at all, because the world models are untrained. All of these policies therefore are trained exclusively on real experiences just like standard PSRO. However, these experiences are also being used to populate the world model's replay buffer. Once the first planning epoch has arrived, computing responses is temporarily paused. The world model is then given a set number of updates to warm-start its parameters, before being used

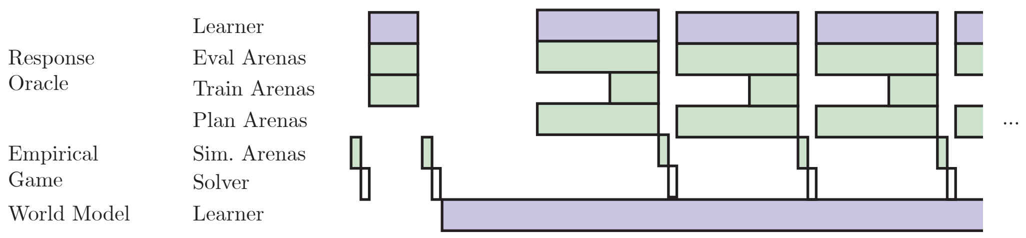


Figure 11.2: **Example Dyna-PSRO process timeline.** In this example, planning is set to occur after the first epoch. Each players' response oracle runs in parallel.

in response calculation. Once the world model's warm-start phase is over, all process proceed concurrently.

Throughout this work, planning is set to begin during the first epoch. Once sufficient data has been generated to train the world model, it's learning coroutine begins. After the world model has completed one million updates, then the response oracles begin using the world model for planning.

11.2 Experimental Setup

Games. Dyna-PSRO is evaluated on three games. The first is the harvest commons game used in the experiments described above, denoted now as "Harvest: Categorical". The other two games come from the MeltingPot [Leibo et al., 2021] evaluation suite and feature rich image-based observations. "Harvest: RGB" is their version of the same commons harvest game, and "Running With Scissors" is a temporally extended version of rock-paper-scissors. Details of both games are provided in Chapter 2.

Experiment. Dyna-PSRO's performance is measured by the quality of the solution it produces when compared against the world-model-free baseline PSRO. The two methods are evaluated on their SumRegret of the combined empirical games (detailed below). We measure SumRegret for intermediate solutions, and report it as a function of the cumulative number of real experiences employed in the respective methods.

Combined-Game Regret. Combined-game regret is an approximate measure of regret that all available estimates to approximate the regret within the true game. Intuitively, combined-game regret is the regret of a strategy with respect to all discovered policies. When comparing empirical-game building algorithms this is formalized as follows:

$$\text{SumRegret}(\boldsymbol{\sigma}, \overline{\boldsymbol{\Pi}}) = \sum_{i \in n} \max_{\pi_i \in \overline{\Pi}_i} \hat{U}_i(\pi_i, \sigma_{-i}) - \hat{U}_i(\sigma_i, \sigma_{-i}), \qquad \overline{\boldsymbol{\Pi}}_i \equiv \bigcup_{\text{method}} \hat{\boldsymbol{\Pi}}_i^{\text{method}}, \tag{11.1}$$

where $\hat{\Pi}$ is the restricted strategy set from one of the algorithms.

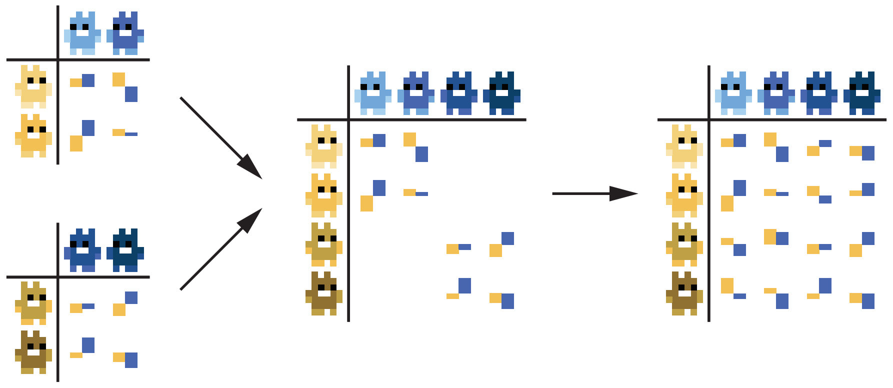


Figure 11.3: **Combined-game construction.** Left: Constituent empirical games. Middle: Combination of the strategy sets and payoff functions. Right: Completion of the empirical game by estimating new strategy profile payoffs.

The process of constructing a combined-game is illustrated in Figure 11.3. Where, the strategy sets (depicted by the toons) across methods are first combined. The new *combined game* that results from this can be initialized with the payoff estimates from the constituent empirical games [Balduzzi et al., 2018]. Unestimated payoffs must then be simulated for the new strategy profiles. Then the complete combined game can be used to compute the combined-game regret from the solutions computed throughout the empirical-game building algorithms.

11.3 Findings

Results. Figure 11.4 presents the results for this experiment. For Harvest: Categorical, Dyna-PSRO found a no regret solution within the combined-game in 3.2×10^6 experiences. Whereas, PSRO achieves a solution of at best 5.45 ± 1.62 within 2×10^7 experiences. In Harvest: RGB, Dyna-PSRO reaches a solution with 0.89 ± 0.74 regret at 5.12×10^6 experiences. At the same time, PSRO had found a solution with 6.42 ± 4.73 regret, and at the end of its run had 2.50 ± 2.24 regret. In the final game, RWS, Dyna-PSRO has $2\text{e}{-3} \pm 5\text{e}{-4}$ regret at 1.06×10^7 experiences, and at a similar point (9.6×10^6 experiences), PSRO has $6.68\text{e}{-3} \pm 2.51\text{e}{-3}$. At the end of the run, PSRO achieves a regret $3.50\text{e}{-3} \pm 7.36\text{e}{-4}$.

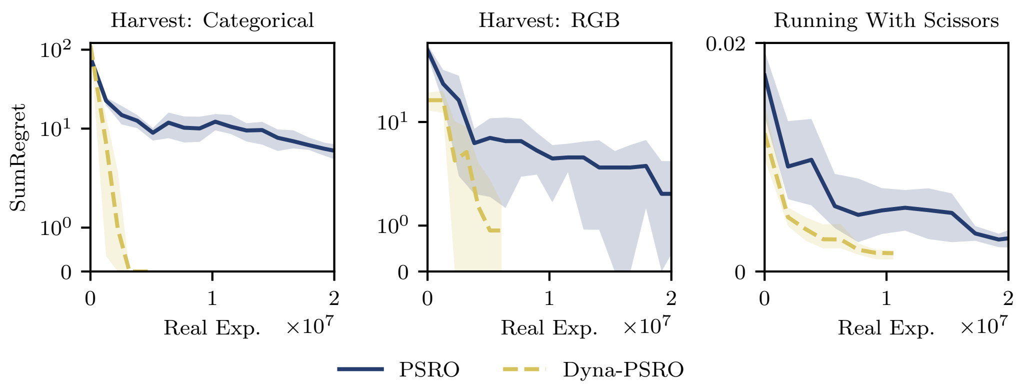


Figure 11.4: **PSRO compared against Dyna-PSRO.** (5 seeds, with $95\,\%$ bootstrapped CI).

Discussion. The results indicate that across all games, Dyna-PSRO consistently outperforms PSRO by achieving a superior solution. Furthermore, this improved performance is realized while consuming fewer real-game experiences. For instance, in the case of Harvest: Categorical, the application of the world model for decision-time planning enables the computation of an effective policy after only a few iterations. On the other hand, we observe a trend of accruing marginal gains in other games, suggesting that the benefits are likely attributed to the transfer of knowledge about the game dynamics. In Harvest: Categorical and Running With Scissors, Dyna-PSRO also had lower variance than PSRO.

Limitations Although our experiments demonstrate benefits for co-learning world models and empirical games, there are several areas for potential improvement. The world models used in this study necessitated observational data from all players for training, and assumed a simultaneous-action game. Future research could consider relaxing these assumptions to accommodate different interaction protocols, a larger number of players, and incomplete data perspectives. Furthermore, our world models functioned directly on agent observations, which made them computationally costly to query. If the generation of experiences is the major limiting factor, as assumed in this study, this approach is acceptable. Nevertheless, reducing computational demands through methods like latent world models presents a promising avenue for future research. Lastly, the evaluation of solution concepts could also be improved. While combined-game regret employs all available estimates in approximating regret, its inherent inaccuracies may lead to misinterpretations of relative performance.

11.4 Conclusion: World Knowledge Transfer

This study showed the mutual benefit of co-learning a world model and empirical game. First, we demonstrated that empirical games provide strategically diverse training data that could inform a more robust world model. We then showed that world models can reduce the computational cost, measured in experiences, of response calculations through planning. These two benefits were combined and realized in a new algorithm, Dyna-PSRO. In our experiments, Dyna-PSRO computed lower-regret solutions than PSRO on several partially observable general-sum games. Dyna-PSRO also required substantially fewer experiences than PSRO, a key algorithmic advantage for settings where collecting experiences is a cost-limiting factor.

Part IV

Conclusion

CHAPTER 12

Conclusion

In this dissertation I investigated the thesis that *[i]n learning-based game-solving algorithms the response learning problems exhibit a common structure that reflects the empirical-game building process, thus facilitating the transfer of knowledge from previous responses and consequently reducing the experiential cost of game solving.* In support of this thesis, I showed how that common structure can be defined across the BR learning problems posed by learning-based game-solving algorithms. I further classified this structure into two key categories: strategic knowledge and world knowledge that can be learned and leveraged. I have presented various methods that utilize this knowledge for transfer learning and efficient game solving.

With all evidence of the thesis in hand, in this final part, I summarize the story so far and chart a path forward. I begin by offering a synopsis of the main contributions made by this dissertation. Following this summary, I turn to the unresolved questions that have emerged from this study or those closely related to it. Finally, I conclude with some closing remarks.

12.1 Summary of Contributions

Our story began by establishing a taxonomy of game-based transferable knowledge. This classification of knowledge is deliberately constructed to reflect the shared structure of the response-learning problems in empirical game building. At its highest level, knowledge is categorized into strategic or world[ly], aligning with the primary parts of this dissertation.

I begin by delving into the role that strategic knowledge plays in game solving. As per our definition, response knowledge is represented by a response policy to fixed coplayers. Assuming the game is fixed, response policies capture the agent's ideal behavior in relation to its strategic context. I frame the primary transfer learning problem faced in game solving with the opponent-mixture transfer problem. In simple terms, this problem revolves around the utilization of pure-strategy responses to generate a response for a mixed strategy. In order to effectively transfer our pure-strategy response knowledge, we need to maintain a belief over which policy our coplayer is currently playing. Opponent-policy belief knowledge forms the other key type of strategic knowledge.

The most general solution to the opponent-mixture transfer problem would be to transfer response policies directly. However, such direct transference is infeasible, as a complete coverage of the state-action space cannot be generally guaranteed. Ensuring such coverage could, at times, considerably compromise the quality of response policies.

Instead of direct policy transfers, we need to employ alternative or supplementary representations of behavior. The Q-value function provides one such representation. It maintains return estimates for all state-action pairs, including those considered sub-optimal within their corresponding responses. It is in this handling of non-optimal behavior that Q-value functions offers advantages over direct policy transfer. In direct policy transfer, non-optimal behavior would require support, potentially undermining the optimality of the response policy. On the other hand, value functions store data about all state-actions, which can be later utilized to derive an optimal policy. Having both optimal and non-optimal behavior available enables transfer to any mixed-strategy response. In light of this, we put forth the Q-Mixing algorithm, which addresses the opponent-mixture transfer problem using Q-values.

The Q-Mixing algorithm operates by taking the mean of the Q-values from response policies for each policy included in the coplayer's mixed strategy. The weight assigned to each component Q-value corresponds to the probability of the coplayer employing that specific pure strategy. In games limited to a single state and given the coplayer's strategy, Q-Mixing can exactly compute a mixed-strategy response. For all other types of games, evidence pertaining to the coplayer's current policy can be gathered during gameplay. This aids an agent in updating their belief regarding the coplayer's policy. In Chapter 6, I delved into methods that aid in constructing an effective model for the likelihood of the opponent's policy. This, in turn, establishes approximate Q-Mixing methods that are practical and demonstrate strong empirical performance.

Having established a viable solution to the opponent-mixture transfer problem, I then turn towards its application in game solving. I introduce two game-solving algorithms that utilize strategic knowledge transfer. A common feature of both algorithms is the modification of the best-response objective, transitioning from responses to mixed-strategy opponents to pure-strategy opponents. This alteration eliminates the opponent sampling process, subsequently reducing the variance in experiences faced by the learner.

The first algorithm, Mixed-Oracles, trains response policies that correspond to opponent individual policies. A response policy to a mixed-strategy opponent is then crafted by combining the response policies corresponding to the individual opponent policies. In this dissertation, I operationalize Q-Mixing to combine pure-strategy responses. However, Mixed-Oracles is not restricted to this; it is broadly applicable to any subroutine capable of combining responses. The exploration of extensions to Q-Mixing or alternative combination-operations present promising avenues for future research.

The second game-solving algorithm that implements strategic knowledge transfer is the Mixed-Opponents algorithm. Mixed-Opponents capitalizes on the additional insight that efficient game solving might necessitate explicit exploration across the policy space. The exploration heuristic it uses posits that non-optimal actions across policies in a mixed strategy suggest an aggregate strategic importance. Bearing this in mind, Mixed-Opponents crafts a new policy that represents an aggregate of the corresponding mixed-strategy opponent. We once again utilize Q-Mixing as our combination method in this context, but like Mixed-Oracles, Mixed-Opponents is not limited to Q-Mixing, and the exploration of alternative combination methods remains an intriguing direction for future research.

Both Mixed-Oracles and Mixed-Opponents provide persuasive evidence that the transfer of strategic knowledge can reduce the computational costs associated with learning-based game-solving algorithms. This forms the initial part of evidence supporting the thesis of this dissertation.

The second part of this dissertation lends further support to the thesis by illustrating how the transfer of world knowledge can lessen the cost of game solving. As game-solving algorithms pertain to a fixed game, the game itself represents a shared structure across each response computation. Consistent with this observation, I define world knowledge as those components of the response learning problem that are independent to strategic variations—thereby, remaining fixed. One representative of world knowledge is the world model, which is a predictive model of the world's transition dynamics and reward function. I explore its potential for reducing the computational cost of game solving through its co-learning alongside an empirical game.

The concurrent learning of a world model and an empirical game offers mutual benefits. World models gain from the broadening of their training data, achieved by incorporating data from across the views of the different strategies that emerge during the building of an empirical game. On the other hand, empirical games can reduce their construction cost by learning and transferring a world model, which aids in training response policies through planning. The improvements for both models were empirically demonstrated by testing their generalization performance to held-out coplayer policies. This investigation ultimately led to the formulation of the Dyna-PSRO algorithm, which effectively harnesses both improvements. Dyna-PSRO learns a world model concurrently with the standard empirical game-building procedures, and as a result, reduces the overall computational cost of game solving.

Dyna-PSRO validates the efficacy of world knowledge transfer when utilized to reduce the cost of game solving. As such, it acts as to capstone the evidence of the second part of the thesis of this dissertation.

12.2 Future Directions & Open Problems

This dissertation only scratches the surface on studying the relationship between transfer learning and game solving. Transfer learning is a rich discipline with a variety of approaches outside of transferring knowledge. In this section, I delineate potential future directions and open problems that either build upon or are strongly correlated with this dissertation. The selected topics represent only a minuscule fraction of the full breadth of problems within this domain, serving merely as a source of inspiration for future researchers.

$>$2 Player Games. The focus of this dissertation is primarily on two-player games, as the algorithms formulated herein are specifically designed for such scenarios. While it is possible to extend these algorithms to many-player games, it is important to note that their complexity grows exponentially with the number of players. As a result, these algorithms become impractical for use in games with many participants. Further research is needed to address this limitation and develop efficient methods for analyzing and solving many-player games.

Strategy Discovery. One potential method to decrease the cost of learning-based game solving involves reducing the number of iterations required for response computation. This can be achieved through effective strategy discovery, as a smaller number of policies need to be utilized in identifying and then capturing the game's solution. The process of strategy discovery necessitates a careful balance between considering a variety of policies (exploration) and fine-tuning already established effective solutions (exploitation) [Smith et al., 2023b]. Most research in strategy discovery has focused on exploration alone [Schvartzman and Wellman, 2009a]. Future research should give due consideration to the explicit treatment of exploration, exploitation, and the interaction between these two aspects.

Alternative Empirical Game Forms. The primary focus of this dissertation is on analyzing transfer learning using the structure of ENFGs. However, there are other empirical game forms that offer opportunities for developing customized transfer learning algorithms by leveraging their unique structures. One promising game form that has recently gained attention is Extensive-Form Empirical Games (EEFGs) [Konicki et al., 2022, McAleer et al., 2021]. EEFGs abstract the temporal structure of a game, which establishes a clear connection with options, a concept that abstracts actions into temporal sub-behaviors [Sutton et al., 1999b]. It is intuitive to consider that the challenges of option discovery and constructing an EEFG are inherently interconnected. Moreover, options can serve as an alternative representation of behavior that can be effectively transferred across similar games. Exploring this connection and its implications could yield valuable insights for future research in this field.

Strategic Planning. The integration of co-learning world models and empirical games opens up several additional opportunities for their joint utilization. Dyna-PSRO demonstrated that planning with a world model can significantly reduce the computational costs associated with game solving. However, Dyna-PSRO's planning approach relied on direct access to the opponent's actions and employed a simple search routine. To further enhance the performance of Dyna-PSRO, more sophisticated planning routines can be explored. An intriguing direction to pursue is incorporating knowledge derived from the empirical game directly into the search routine. This integration has the potential to improve performance during the learning phase as well as when deploying the solution after solving the game [Li et al., 2023]. Future research could investigate and develop techniques that effectively leverage empirical game knowledge within the search routine, building upon the foundations established by Dyna-PSRO.

World Model Extensions. In this dissertation, a deterministic representation was adopted to model a potentially stochastic game as part of the world modeling approach. To improve the accuracy of world knowledge, incorporating more advanced world modeling technologies, such as those derived from single-agent RL, would be beneficial. This advancement is expected to enhance the effectiveness of the transfer process. Additionally, investigating the interplay between different modeling approaches and different classes of games can offer further advantages. Currently, our world model architecture assumes a simultaneous action game, which limits its applicability to games with general interaction procedures. Extending the world model to accommodate various interaction procedures presents a significant challenge in terms of modeling and learning. Exploring these aspects in future research holds the potential for significant advancements in the field. By integrating more advanced world modeling techniques and exploring the interactions between different modeling approaches and game classes, a more accurate and adaptable representation of world knowledge can be achieved.

Cross-Empirical-Game Transfer. While this dissertation primarily focuses on transfer learning within the context of reinforcement learning, it is worth noting that transfer learning can also be applied to the construction of empirical games themselves. Solutions or partial solutions obtained from similar or identical games can be transferred and utilized to expedite the process of solving a new game (e.g., parameterized family of games [Gatchel and Wiedenbeck, 2023]). Additionally, parallel game-solving procedures applied to similar games can exchange learned knowledge and collectively enhance each other's performance. To demonstrate the potential of this approach, let's consider the role of an auctioneer who employs empirical game analysis to inform mechanism design. Suppose the auctioneer has already utilized an empirical game to guide the creation and deployment of an auction mechanism. If the auctioneer wishes to modify the mechanism, it would typically require constructing a new empirical game. However, due to the shared nature of auctions, there may exist opportunities to transfer insights from the original empirical game regarding the strategy space of empirical games. This transfer of knowledge could make the process more efficient and informative, facilitating the modification of the auction mechanism. By exploring the application of transfer learning to empirical game construction, researchers and practitioners can leverage existing knowledge and solutions from similar games to accelerate the process of solving new games, leading to improved efficiency and effectiveness in mechanism design and other related areas.

Alternative Transfer Methods. Transfer learning is a diverse and multidisciplinary field that encompasses various approaches and perspectives influenced by numerous other disciplines. In this dissertation, the focus was primarily on one specific type of transfer learning, namely the transfer of knowledge. However, depending on the task dissimilarities, there is a wide range of other transfer methodologies that could be explored. These alternative transfer methods include task mappings [Taylor et al., 2007, Fernández and Veloso, 2006], reward shaping, optimization or learner state transfer, and task selection mechanisms. These approaches can be applied at different stages, whether at the level of reinforcement learning, during the construction of empirical games, or at some intermediate point in the process. By considering these different transfer methods, a more comprehensive and flexible framework for game solving can be established. For a comprehensive review of transfer learning methods that can be potentially adapted for game solving, I recommend referring to the literature review by Taylor and Stone [2009].

12.3 Concluding Remarks

I started this dissertation with the observation that game-solving algorithms often treat learning in different strategic contexts as independent problems. Naturally, this served as motivation for the inclusion of transfer learning in game solving. It also highlights a common trend in game-solving and reinforcement-learning algorithms: the tendency to treat problems in isolation. While isolating problems can simplify them and focus learning on critical areas, this approach does not fully reflect reality.

However, in reality, games are not so conveniently isolated from the rest of the world. Human players carry a lifetime of experiences that influence their approach to each gameplay [Camerer, 2003, Haruvy and Stahl, 2004, Ho et al., 2007]. Whether it's derived from similar games or past encounters with the same game, humans' inherent ability to transfer knowledge enables them to swiftly adapt to new games or coplayers.

The design of our AI agents and their learning algorithms should take into account both the game-focused perspective and the broader, lifelong perspective. Maintaining these dual viewpoints allows us to better comprehend the efficiency gap between AI and humans, and take strides towards reducing it. This isn't a novel observation, but rather, a crucial thread in the quilt of our AI development, which is often overshadowed. As our world becomes increasingly interconnected, the efficient strategic reasoning that transfer learning affords will continue to rise in importance. I hope this dissertation can serve as a valuable resource, its ideas transferred, reimagined, and applied to the multitude of unresolved problems that still lie ahead.

BIBLIOGRAPHY

Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

Haitham Bou Ammar, Eric Eaton, José Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *24th International Conference on Artificial Intelligence*, pages 3345–3349, 2015.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-DQN: variance reduction and stabilization for deep reinforcement learning. In *34th International Conference on Machine Learning*, pages 176–185, 2017.

David Balduzzi, Karl Tuyls, Julien Pérolat, and Thore Graepel. Re-evaluating evaluation. In *32nd Conference on Neural Information Processing Systems*, 2018.

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech M. Czarnecki, Julien Pérolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *36th International Conference on Machine Learning*, 2019.

Philip Ball, Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Ready policy one: World building through active learning. In *37th International Conference of Machine Learning*, 2020.

Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *20th International Joint Conference on Artificial Intelligence*, pages 672–677, 2007.

Nolan Bard, Michael Johanson, Neil Burch, and Michael Bowling. Online implicit agent modelling. In *12th International Conference on Autonomous Agents and Multiagent Systems*, pages 255–262, 2013.

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 280, 2020.

Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *29th AAAI Conference on Artificial Intelligence*, pages 2010–2016, 2015.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. doi: 10.1613/jair.3912.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *28th Conference on Neural Information Processing Systems*, pages 1171–1179, 2015.

Avrim Blum and Yishay Mansour. Learning, regret minimization, and equilibria. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 79–101. 2007.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

George W Brown. Iterative solution of games by fictitious play. In *Activity analysis of production and allocation*, volume 13, pages 374–376, 1951.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541, 2006.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *22nd Conference on Neural Information Processing Systems*, 2018.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

C.F. Camerer. *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press, 2003.

Rich Caruana. Multitask learning. *Machine Learing*, 28(1):41–75, 1997.

Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, and Manuel Kroiss. Reverb: A framework for experience replay, 2021.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *5th International Conference on Learning Representations*, 2017.

Valliappa Chockingam, Tegg Taekyong Sung, Feryal Behbanai, Rishab Gargeya, Amlesh Sivanantham, and Aleksandra Malysheva. Extending world models for multi-agent reinforcement learning in malmö. In *Joint AIIDE 2018 Workshops co-located with the 14th AAAI conference on artificial intelligence and interactive digital entertainment*, 2018.

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *15th National Conference on Artificial Intelligence*, pages 746–752, 1998.

Brian Collins. Combining opponent modeling and model-based reinforcement learning in a two-player competitive game. Master's thesis, University of Edinburgh, 2007.

Jacob W. Crandall. Just add pepper: Extending learning algorithms for repeated matrix games to repeated Markov games. In *11th International Conference on Autonomous Agents and Multiagent Systems*, pages 399–406, 2012.

Wojciech Czarnecki, Siddhant Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Nicolas Heess, Simon Osindero, and Razvan Pascanu. Mix & match agent curricula for reinforcement learning. In *35th International Conference on Machine Learning*, pages 1087–1095, 2018.

B. Curtis Eaves. The linear complementarity problem. *Management Science*, 17(9):612–634, 1971.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *35th International Conference on Machine Learning*, 2018.

Fernando Fernández and Manuela Veloso. Policy reuse for transfer learning across tasks with different state and action spaces. In *ICML'06 Workshop on Structural Knowledge Transfer for ML*, 2006.

Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *34th International Conference on Machine Learning*, pages 1146–1155, 2017.

Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130, 2018a.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *32nd AAAI Conference on Artificial Intelligence*, pages 2974–2982, 2018b.

David Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49 (2):325–346, 2002.

Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136, 1975.

Sam Ganzfried and Tuomas Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *10th International Conference on Autonomous Agents and Multiagent Systems*, 2011.

Madelyn Gatchel and Bryce Wiedenbeck. Learning parameterized families of games. In *22nd International Conference on Autonomous Agents and Multiagent Systems*, pages 1044–1052, 2023.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In *36th International Conference on Machine Learning*, pages 2170–2179, 2019.

Peter W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990. doi: 10.1145/84544.84551.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, volume 1. MIT Press, 2016.

Amy Greenwald and Keith Hall. Correlated-Q learning. In *20th International Conference on Machine Learning*, pages 242–249, 2003.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *14th International Conference on Neural Information Processing Systems*, pages 1523–1530, 2001.

Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *16th International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83, 2017.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *31st Conference on Neural Information Processing Systems*, 2018a.

David Ha and Jürgen Schmidhuber. World models. In *arXiv preprint arXiv:1803.10122*, 2018b.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68:1127–1150, 2000.

Ernan Haruvy and Dale O. Stahl. Deductive versus inductive equilibrium selection: experimental results. *Journal of Economic Behavior & Organization*, 53(3):319–331, 2004.

He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *33rd International Conference on Machine Learning*, pages 1804–1813, 2016.

Johannes Heinrich. *Reinforcement Learning from Self-Play in Imperfect-Information Games*. PhD thesis, University College London, 2017.

Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *32nd International Conference on Machine Learning*, 2015.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Pablo Hernandez-Leal and Michael Kaisers. Learning against sequential opponents in repeated stochastic games. In *3rd Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2017a.

Pablo Hernandez-Leal and Michael Kaisers. Towards a fast detection of opponents in repeated stochastic games. In *16th International Conference on Autonomous Agents and Multiagent Systems*, pages 239–257, 2017b.

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

Todd Hester and Peter Stone. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. In *Machine Learning for Robotics (MLR)*, 2012.

Geoffrey Hinton. Coursera neural networks for machine learning lecture 6. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2018. Accessed: 2023-04-18.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS: Deep Learning and Representation Learning Workshop*, 2014.

Teck H. Ho, Colin F Camerer, and Juin-Kuan Chong. Self-tuning experience weighted attraction learning in games. *Journal of Economic Theory*, 133(1):177–198, 2007.

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

Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, Léonard Hussenot, Robert Dadashi, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

G. Zacharias Holland, Erin Talvitie, and Michael Bowling. The effect of planning shape on dyna-style planning in high-dimensional state spaces. In *FAIM workshop "Prediction and Generative Modeling in Reinforcement Learning"*, 2018.

HumanCompatibleAI. https://github.com/HumanCompatibleAI/multi-agent, 2019.

Pararawendy Indarjo. Deep state-space models in multi-agent systems. Master's thesis, Leiden University, 2019.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computing*, 3(1):79–87, 1991.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Marco A. Janssen, Robert Holahan, Allen Lee, and Elinor Ostrom. Lab experiments for the study of social-ecological systems. *Science*, 328(5978):613–617, 2010. doi: 10.1126/science.1183532. URL https://www.science.org/doi/abs/10.1126/science.1183532.

Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In *30th International Conference on Neural Information Processing Systems*, 2007.

Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *19th International Joint Conference on Artificial Intelligence*, pages 752–757, 2005.

Patrick R. Jordan, L. Julian Schvartzman, and Michael P. Wellman. Strategy exploration in empirical games. In *9th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1131–1138, 2010.

Leslie Pack Kaelbling. Learning to achieve goals. In *13th International Joint Conference on Artificial Intelligence*, pages 1094–1099, 1993.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, 2015.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning*, pages 282–293. Springer, 2006.

Daphne Koller and Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4:528–552, 1992.

Christine Konicki, Mithun Chakraborty, and Michael P. Wellman. Exploiting extensive-form structure in empirical game-theoretic analysis. In *Web and Internet Economics: 18th International Conference*, 2022.

George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *23rd International Conference on Machine Learning*, pages 489–496, 2006.

Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Orr Krupnik, Igor Mordatch, and Aviv Tamar. Multi-agent reinforcement learning with multi-step generative models. In *4th Conference on Robot Learning*, pages 776–790, 2020.

Harold W. Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games*, 2(28):193–216, 1953.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *6th International Conference on Learning Representations*, 2018.

Andrew K. Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. In *7th International Conference on Learning Representations*, 2019.

Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *31st Conference on Neural Information Processing Systems*, page 4193–4206, 2017.

Marc Lanctot, John Schultz, Neil Burch, Max Olan Smith, Daniel Hennes, Thomas Anthony, and Julien Perolat. Population-based evaluation in repeated rock-paper-scissors as a benchmark for multiagent reinforcement learning, 2023.

Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *16th International Conference on Autonomous Agents and Multiagent Systems*, 2017.

Joel Z. Leibo, Edgar Duéñez-Guzmán, Alexander Sasha Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charles Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. In *38th International Conference on Machine Learning*, pages 6187–6199, 2021.

Zun Li, Marc Lanctot, Kevin McKee, Luke Marris, Ian Gemp, Daniel Hennes, Paul Muller, Kate Larson, Yoram Bachrach, and Michael P. Wellman. Search-improved game-theoretic multiagent reinforcement learning in general and negotiation games (extended abstract). In *32nd International Conference on Autonomous Agents and Multiagent Systems*, 2023.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992. doi: 10.1007/bf00992699.

Alan J. Lockett, Charles L. Chen, and Risto Miikkulainen. Evolving explicit opponent models in game playing. In *9th Annual Genetic and Evolutionary Computational Conference*, 2007.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *31st Conference on Neural Information Processing Systems*, pages 6382–6393, 2017.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: Multi-agent variational exploration. In *33rd Conference on Neural Information Processing Systems*, 2019.

A. A. Markov. *Theory of Algorithms*. Academy of Sciences of the USSR, 1954. Translation of Works of the Mathematical Institute, Academy of Sciences of the USSR, v. 42. Original title: Teoriya algorifmov. [QA248.M2943 Dartmouth College library. U.S. Dept. of Commerce, Office of Technical Services, number OTS 60-51085.].

Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. ICML, pages 7480–7491, 2021.

Stephen McAleer, John Lanier, Kevin Wang, Pierre Baldi, and Roy Fox. XDO: A double oracle algorithm for extensive-form games. In *35th Conference on Neural Information Processing Systems*, 2021.

Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory. http://www.gambit-project.org/, 2016.

H. Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *20th International Conference on Machine Learning*, pages 536–543, 2003.

Richard Mealing and Jonathan L Shapiro. Opponent modeling by expectation–maximization and sequence prediction in simplified poker. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 9, pages 11–24, 2015.

Vicent Michalski, Roland Memisevic, and Kishore Konda. Modeling deep temporal dependencies with recurrent grammar cells. In *27th Conference on Neural Information Processing Systems*, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14(1): 124–143, 1996.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13(1):103–130, 1993.

Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Pérolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Remi Munos. A generalized training approach for multiagent learning. In *8th International Conference on Learning Representations*, 2020.

John Forbes Nash Jr. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

Johan S. Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *38th International Conference on Machine Learning*, 2021.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. In *28th Conference on Neural Information Processing Systems*, 2015.

Junhyuk Oh, Satinderg Singh, and Honglak Lee. Value prediction network. In *30th Conference on Neural Information Processing Systems*, pages 6118–6128, 2017.

Frans A. Oliehoek. Decentralized pomdps. In *Reinforcement Learning*, volume 12, pages 471–503. 2012.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *34th International Conference on Machine Learning*, 2017.

Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Pérolat, and Remi Munos. α-rank: Multi-agent evaluation by evolution. *Scientific Reports*, 9, 2019.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

S. Phelps, M. Marcinkiewicz, and S. Parsons. A novel method for automatic strategy acquisition in N-player non-zero-sum games. In *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, page 705–712, 2006.

Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Julien Pérolat, Joel Z. Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *31st Conference on Neural Information Processing Systems*, 2017.

Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. In *35th International Conference on Machine Learning*, 2018.

A. Rapoport and A. M. Chammah. *Prisoner's Dilemma*. University of Michigan Press, 1965.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *35th International Conference on Machine Learning*, pages 4295–4304, 2018.

Mark Ring. Representing knowledge as predictions (and state as knowledge). *arXiv preprint arXiv:2112.06336*, 2013.

Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. Bayesian policy reuse. *Machine Learning*, 104:99–127, 2016.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

Andrei A. Rusu, Sergio Gómez Colmenarejo, Çağlar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation, 2015.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

L. Julian Schvartzman and Michael P. Wellman. Exploring large strategy spaces in empirical game modeling. In *AAMAS-09 Workshop on Agent-Mediated Electronic Commerce*, 2009a.

L. Julian Schvartzman and Michael P. Wellman. Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning. In *8th International Conference on Autonomous Agents and Multiagent Systems*, pages 249–256, 2009b.

Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Angieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for contiual learning. In *35th International Conference on Machine Learning*, 2018.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *37th International Conference of Machine Learning*, pages 8583–8592, 2020.

Macheng Shen and Jonathan P. How. Robust opponent modeling via adversarial ensemble reinforcement learning. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling*, volume 31 of *Special Track on Planning and Learning*, 2021.

Felipe Silva and Anna Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017a.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *34th International Conference on Machine Learning*, volume 70, pages 3191–3199, 2017b.

Max Olan Smith and Michael P. Wellman. Co-learning empirical games and world models. *CoRR*, 2023. URL https://arxiv.org/abs/2305.14223.

Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Iterative empirical game solving via single policy best response. In *9th International Conference on Learning Representations*, 2021.

Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Learning to play against any mixture of opponents. *Frontiers in Artificial Intelligence*, page to appear, 2023a.

Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Strategic knowledge transfer. *Journal of Machine Learning Research*, 24:to appear, 2023b.

Matthijs Snel and Shimon Whiteson. Learning potential functions and their representations for multi-task reinforcement learning. In *13th International Conference on Autonomous Agents and Multiagent Systems*, pages 637–681, 2014.

Chen Sun, Per Karlsson, Jiajun Wu, Joshua B Tenenbaum, and Kevin Murphy. Stochastic prediction of multi-agent interactions from partial observations. In *7th International Conference on Learning Representations*, 2019.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *17th International Conference on Autonomous Agents and Multiagent Systems*, 2018.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *7th International Workshop on Machine Learning*, pages 216–224, 1990.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063, 1999a.

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999b.

Richard S Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael P. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *28th Conference on Uncertainty in Artificial Intelligence*, 2012.

Erin Talvitie. Model regularization for stable sample rollouts. In *30th Conference on Uncertainty in Artificial Intelligence*, 2014.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *10th International Conference on Machine Learning*, pages 330–337, 1993.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Value functions for RL-based behavior transfer: A comparative study. In *20th National Conference on Artificial Intelligence*, pages 880–885, 2005.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

Peter D. Taylor and Leo B. Jonker. Evolutionarily stable strategies and game dynamics. *Mathematical Biosciences*, 40:146–156, 1978.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38 (3):58–68, 1995.

Gerald Tesauro. Extending Q-learning to general adaptive multi-agent systems. In *16th International Conference on Neural Information Processing Systems*, pages 871–878, 2003.

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, 1995.

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*, 2019.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997. doi: 10.1109/tac.1997.585906.

A. Tucker. A two-person dilemma. *UMAP Journal*, 2(1), 1950.

Karl Tuyls, Julien Pérolat, Marc Lanctot, Edward Hughes, Richard Everett, Joel Z. Leibo, Csaba Szepesvári, and Thore Graepel. Bounds and dynamics for empirical game theoretic analysis. *Autonomous Agents and Multi-Agent Systems*, 34(7), 2020.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *30th AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.

Alexander Sasha Vezhnevets, Yuhuai Wu, Remi Leblond, and Joel Z. Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent reinforcement learning. In *37th International Conference on Machine Learning*, pages 9733–9742, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z.

Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(50):220–246, 1996.

Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

Thomas J. Walsh, Lihong Li, and Michael L. Littman. Transferring state abstractions between MDPs. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

William Walsh, Rajarshi Das, Gerald Tesauro, and Jeffrey Kephart. Analyzing complex strategic interactions in multi-agent systems. In *AAAI-02 Workshop on Game Theoretic and Decision Theoretic Agents*, 2002.

Xiaofeng Wang and Tuomas Sandholm. Learning near-Pareto-optimal conventions in polynomial time. In *16th International Conference on Neural Information Processing Systems*, pages 863–870, 2003.

Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. In *33rd AAAI Conference on Artificial Intelligence*, AAAI, pages 1401–1408, 2019.

Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *28th International Conference on Neural Information Processing Systems*, pages 2746–2754, 2015.

Michael P. Wellman. Methods for empirical game-theoretic analysis. In *21st National Conference on Artificial Intelligence*, page 1552–1555, 2006.

Daniël Willemsen, Mario Coppola, and Guido CHE de Croon. MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. doi: 10.1023/a:1022676722315.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2), 1989.

Mason Wright, Yongzhao Wang, and Michael P. Wellman. Iterated deep reinforcement learning in games: History-aware training for improved stability. In *20th ACM Conference on Economics and Computation*, EC, pages 617–636, 2019.

Fan Yang, Gabriel Barth-Maron, Piotr Stańczyk, Matthew Hoffman, Siqi Liu, Manuel Kroiss, Aedan Pope, and Alban Rrustemi. Launchpad: A programming model for distributed machine learning research. *arXiv preprint arXiv:2106.04516*, 2021a. URL https://arxiv.org/abs/2106.04516.

Tainpei Yang, Weixun Wang, Hongyao Tang, Jianye Hao, Zhaopeng Meng, Hangyu Mao, Dong Li, Wulong Liu, Chengwei Zhang, Yujing Hu, Yingfeng Chen, and Changjie Fan. An efficient transfer learning framework for multiagent reinforcement learning. In *35th Conference on Neural Information Processing Systems*, 2021b.

Tianpei Yang, Jianye Hao, Zhaopeng Meng, Chongjie Zhang, Yan Zheng, and Ze Zheng. Towards efficient detection and optimal response against sophisticated opponents. In *28th International Joint Conference on Artificial Intelligence*, pages 623–629, 2019a.

Tianpei Yang, Jianye Hao, Zhaopeng Meng, Yan Zheng, Chongjie Zhang, and Ze Zheng. Bayes-ToMoP: A fast detection and best response algorithm towards sophisticated opponents. In *19th International Conference on Autonomous Agents and Multiagent Systems*, 2019b.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. In *33rd Conference on Neural Information Processing Systems*, 2020.

Kaiqing Zhang, Sham Kakade, Tamer Basar, and Lin Yang. Model-based multi-agent RL in zero-sum markov games with near-optimal sample complexity. In *33rd Conference on Neural Information Processing Systems*, 2020.

Qizhen Zhang, Chris Lu, Animesh Garg, and Jakob Foerster. Centralized model and exploration policy for multi-agent RL. In *21st International Conference on Autonomous Agents and Multiagent Systems*, 2022.

Yan Zheng, Zhaopeng Meng, Jianye Hao, Zongzhang Zhang, Tianpei Yang, and Changjie Fan. A deep Bayesian policy reuse approach against non-stationary agents. In *31st International Conference on Neural Information Processing Systems*, 2018.