Advances in Meta-Reinforcement Learning and Imitation Learning

Risto Vuorio

Keble College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2024

Abstract

Recent advances in machine learning (ML) have led to remarkable progress in artificial intelligence (AI), enabling computers to solve previously intractable problems. While breakthroughs in generative AI, such as large language models and diffusion-based image generators, have captured public attention, the development of interactive AI agents remains an emerging frontier. Unlike generative models, interactive agents are designed to learn from their interactions with dynamic environments, enabling applications such as robotics, autonomous driving, and adaptive control. This thesis explores two critical aspects of learning for interactive AI: meta-reinforcement learning (meta-RL) and imitation learning (IL).

In the first part, we address the problem of improving the sample efficiency of reinforcement learning (RL) algorithms through meta-learning. By developing novel meta-RL methods, we enable agents to adaptively learn how to learn, enhancing their ability to assign credit and optimize behavior efficiently. We introduce a meta-gradient-based approach to adaptively assign credit over time and present theoretical and empirical insights into the challenges of meta-gradient estimation. The second part of the thesis focuses on IL in settings where the expert and imitator have different observations of the environment, leading to what is known as the "imitation gap". We propose algorithms to tackle this gap, modeling the missing information as confounding latent variables or Bayesian priors, and show that these methods enable effective imitation in complex scenarios.

Overall, this thesis contributes to the development of interactive learning agents through advances in meta-RL and IL, providing algorithms, empirical analysis, and theoretical insights. Our work not only advances the efficiency and adaptability of learning agents, but also lays the groundwork for future research to build more robust, safe, and generalizable AI systems capable of interacting effectively with complex environments.

Advances in Meta-Reinforcement Learning and Imitation Learning

Risto Vuorio

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2024

Acknowledgements

First, I extend my deepest gratitude to my supervisor, Shimon Whiteson. His guidance has been invaluable throughout my DPhil journey. Shimon not only took a chance on me by welcoming me into the program but also greatly improved me as a scientist. His insight and advice were crucial whenever I faced challenges or uncertainty about the direction of my research.

I thank Scatcherd European Scolarship and Engineering and Physical Sciences Research Council Studentship for funding my DPhil program. Furthermore, I thank Keble College and the Department of Computer Science for supporting me in my research and other activities.

I am grateful for the collaboration and support of my co-authors Jacob Beck, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, Zeyu Zheng, Richard Lewis, Satinder Singh, Gregory Farquhar, Jakob Foerster, Pim de Haan, Johann Brehmer, Hanno Ackermann, Daniel Dijkman, Taco Cohen, Chris Lu, Minqi Jiang, Jack Parker-Holder, Mattie Fellows, Cong Lu, and Clémence Grislain. Your contributions and insights have played a pivotal role in shaping this thesis, and I am grateful for the knowledge, guidance, and camaraderie that we have shared.

A special thanks goes to the team at Qualcomm Research Amsterdam, where I had the opportunity to conduct a fruitful and rewarding internship. I am especially grateful to my supervisors, Taco Cohen and Daniel Dijkman, for their guidance, and to my collaborators, Pim de Haan, Johann Brehmer, and Hanno Ackermann, who I could not have finished the project without. This internship experience not only led to a published paper, but also contributed a chapter to this thesis. Another result of the internship was the personal discovery of the great city of Amsterdam, which I hope to visit many times in the future.

I am indebted to the Waymo Research Oxford team, where I had another enriching internship experience. I thank my host, Nico Montali, for teaching me so much about building and benchmarking machine learning systems at scale. The tightly knit team in the Oxford office made my time there both enjoyable and highly educational. I learned a lot from others at Waymo, and I am particularly grateful to Yiren Lu and Chiyu "Max" Jiang for their evaluation and feedback on my work.

The camaraderie and friendship within the WhiRL lab were essential to maintaining morale. I would not have made it without the support and encouragement

of my lab mates: Alex Goldie, Alex Zakharov, Matthew Jackson, Zheng Xiong, Nasma Dasser, Benjamin Ellis, Jacob Beck, Kristian Hartikainen, Tarun Gupta, Jelena Luketina, Matt Smith, Charline Le Lan, Vitaly Kurin, Anuj Mahajan, Luisa Zintgraf, Shangtong Zhang, Tabish Rashid, Christian Schroeder de Witt, Maximilian Igl, Mattie Fellows, Mingfei Sun, Bei Peng, and Wendelin Boehmer. Thank you for the insightful discussions, the laughter, and the shared journey.

I want to thank all of my friends who have stuck with me throughout my DPhil even when it has meant me being a semi-hermit at times: Kristian Hartikainen, Kenneth Blomqvist, Dann Mensah, Ats Nisov, Pyry Haulos and family, Antti Kemppainen, Heikki Leskelä, Valtteri Töysä, Karhen Pertsat, Tuolinkantajat, Juuso Haavisto, Vivek Veeriah, Chris Grimm, Ethan Brooks, Ondrej Bajgar, Chris Lu, Timon Willi, Andrei Lupu, Ben Ellis, everyone from FLAIR, Kira Mulcahy, Christopher Kuhl, Ylias Saily, Edward Shellard, Parkrunners from Keble, Kylie, Katie, Hanson, Anna, and Paljana from HBAC, John, Alex, and Arthur from Iffley Road, and the many others whose friendship I have been fortunate to cherish along the way.

Lastly, I am deeply grateful to my family for their unwavering support throughout my DPhil. To my mom, dad, and my brothers Markus and Antti, along with their respective families — your love, encouragement, and belief in me have been my anchor throughout this journey. I could not have done this without you.

Thank you for being part of this incredible chapter of my life.

Abstract

Recent advances in machine learning (ML) have led to remarkable progress in artificial intelligence (AI), enabling computers to solve previously intractable problems. While breakthroughs in generative AI, such as large language models and diffusion-based image generators, have captured public attention, the development of interactive AI agents remains an emerging frontier. Unlike generative models, interactive agents are designed to learn from their interactions with dynamic environments, enabling applications such as robotics, autonomous driving, and adaptive control. This thesis explores two critical aspects of learning for interactive AI: meta-reinforcement learning (meta-RL) and imitation learning (IL).

In the first part, we address the problem of improving the sample efficiency of reinforcement learning (RL) algorithms through meta-learning. By developing novel meta-RL methods, we enable agents to adaptively learn how to learn, enhancing their ability to assign credit and optimize behavior efficiently. We introduce a meta-gradient-based approach to adaptively assign credit over time and present theoretical and empirical insights into the challenges of meta-gradient estimation. The second part of the thesis focuses on IL in settings where the expert and imitator have different observations of the environment, leading to what is known as the "imitation gap". We propose algorithms to tackle this gap, modeling the missing information as confounding latent variables or Bayesian priors, and show that these methods enable effective imitation in complex scenarios.

Overall, this thesis contributes to the development of interactive learning agents through advances in meta-RL and IL, providing algorithms, empirical analysis, and theoretical insights. Our work not only advances the efficiency and adaptability of learning agents, but also lays the groundwork for future research to build more robust, safe, and generalizable AI systems capable of interacting effectively with complex environments.

Contents

1 Introduction

We live in a very exciting time for artificial intelligence (AI) research. Recent advances in machine learning (ML), which is a prominent subfield of AI, have enabled novel ways to use computers to solve previously intractable problems. Among the most captivating examples are human-like chat made possible by large language models [Achiam et al., 2023, Brown, 2020], as well as photorealistic images [Ramesh et al., 2022, Saharia et al., 2022] and lifelike videos [Ho et al., 2022] generated by diffusion models [Sohl-Dickstein et al., 2015]. The central drivers behind these advances have been the massive increases in computing power dedicated to training ML models and the ever-growing size of the datasets used in training [Hoffmann et al., 2022]. As thrilling as these advances are, they mostly represent the great ability of ML models known as neural networks to fit arbitrary data distributions and *to generate new examples of those data.* In contrast, we are still in the early stages of enabling a more interactive AI, centered on *agents*, which can, for example, control robots, safely drive vehicles, and adapt to novel situations regardless of the setting. The relatively slower progress in training these agents suggests that it requires going beyond the fitting of static data distributions. A central topic of research for doing that is learning from the interactions of the agents with their environments themselves. Although the applications of ML in these interactive settings have not impacted the public perception of AI quite as

much as generative AI has, significant advances have been made. Among the most impressive are the game-playing agents that beat the best players in the game of go and chess [Silver et al., 2016, 2017] and the agents that allow more flexible control of legged robots [Lee et al., 2020]. Furthermore, the algorithms that power these interactive agents are now used to push the generative AI even further by interactively fine-tuning the models [Achiam et al., 2023, Christiano et al., 2017]. In this thesis, we present advances in algorithms for learning these interactive agents in the fields of meta-reinforcement learning (meta-RL) and imitation learning (IL).

This thesis is organized into two distinct but complementary parts, focusing on meta-RL and the imitation gap problem in IL. While meta-RL and IL are both approaches for sample efficient learning of interactive AI agents, they address different facets of the problem. On the one hand, meta-RL focuses on enhancing the sample efficiency of RL by enabling agents to "learn how to learn" through their experiences, adapting their learning strategies to maximize rewards over time. This is crucial in settings where the progress on the task can be easily evaluated, and therefore the rewards for the agents are clear. On the other hand, IL deals with scenarios where providing expert demonstrations of the task is more feasible than specifying complex reward structures. Specifically, we focus on the imitation gap problem, where discrepancies between the expert's and imitator's observations lead to challenges in directly replicating expert behaviors. While meta-RL is an important direction in the long term, when AI agents increasingly deal with tasks for which human experts cannot be found, in the short term ML is dominated by fitting static task distributions such as demonstrations of expert data in IL. Therefore, we split the work into two parts, aiming to contribute to both the short-term and long-term progress in AI. Next, we discuss these problem settings in detail and our contributions to them.

1.1 Meta-Reinforcement Learning

Reinforcement learning (RL) is a field that studies how to train agents that interact with the environment and maximize the *rewards* they collect during the interaction.

As an example of an RL problem, consider a simulated traffic scenario as the environment in which the agent takes *actions* for controlling a car. We want to train an agent that safely follows a sequence of waypoints while adhering to rules and cultural norms of traffic. To guide the agent toward the desired behavior, i.e., having it choose the best actions in each situation, we have to define a *reward function*. This function rewards the agent for achieving waypoints and participating in the traffic in a social way, but also penalizes it for dangerous behavior. Then we start the simulation and have the agent choose the actions that control the vehicle. After we see the outcome of the simulation, we run an optimization algorithm to change the agent's behavior to collect as much reward as possible. In principle, this approach can be used to solve any problem for which a reward function can be defined [Sutton and Barto, 2018]. In practice, it requires extremely many samples to find a *policy*, which maximizes the rewards and thus achieves our desired goals. In this thesis, we examine methods that improve the sample efficiency of RL algorithms. More specifically, we consider algorithms that adapt during the data collection to improve their sample efficiency. This process of *learning how to learn* is known as meta-learning and is an active area of research in machine learning [Hospedales et al., 2020, Huisman et al., 2021, Vanschoren, 2018]. Our topic, meta-learning specifically RL algorithms, is known as meta-RL [Beck et al., 2023, Schmidhuber, 1987, Schmidhuber et al., 1997, Thrun and Pratt, 1998].

Before moving onto meta-RL, let's first consider how an RL algorithm learns from the interactions to see why RL is so data inefficient. In RL, the interaction between the agent and the environment consists of the agent choosing actions in different states of the environment and the environment progressing to new states in response to the actions. The reward function computes the rewards based on the goodness of the states the agent reaches. In order to maximize the rewards, the agent has to explore the different states of the environment. In the simulated driving example above, the states consist of the locations, velocities, and accelerations of all the traffic participants, as well as the locations and attributes of the lanes and other static objects that define the static driving scenario. In each state, the agent may

have hundreds of actions to choose from. The actions may also be continuous. Due to the continuous simulation of physics, there is an infinite number of states the agent may be in. It is therefore infeasible for the agent to exhaustively visit all states to see what rewards it may get. Furthermore, since we care about driving behavior throughout the interaction, we are not looking for the single most rewarding state the agent may reach, but instead maximizing the sum of rewards the agent incurs, also known as the *return*. To learn a policy that maximizes the return, the agent has to learn to recognize which states are similar to each other, choose which states are worth exploring further, and how to accurately assign credit for the rewards it receives in the future to the actions it took in the past. Therefore, solving the RL problem is much harder than a simple search over the state-action space and requires a large number of samples to first accurately estimate the returns and then learn a policy which maximizes them.

A popular category of RL algorithms that tackle these problems is known as policy gradient methods. Policy gradient methods directly model the policy, i.e., the probability distribution of the actions the agent takes in each state, with a function approximator such as a neural network. Given some estimate of the return, the policy gradient methods do stochastic gradient ascent on the policy parameters to maximize the probability of choosing actions that maximize the return. Due to the intractable number of possible *trajectories* the interaction may take, policy gradient methods need to rely on many approximations of the returns and other quantities involved in the optimization. A sample-based Monte Carlo estimate of the return works in theory but has a high variance in practice, which further increases the sample complexity of learning a good policy. To reduce the variance and, therefore, lower the sample complexity, the user can choose return estimates that trade off variance for increased bias. One option is to choose a different credit-assignment heuristic, which does not treat all future rewards equally. Different credit assignment heuristics have been studied in the history of RL [Schulman et al., 2015, Sutton, 1988], but a single best way to do it throughout the training of an agent has not emerged.

To develop a more sample efficient RL algorithm, instead of manually choosing a static credit assignment heuristic, in chapter 4, we propose a meta-RL method to adapt the heuristic during learning. This has the advantage that the credit can be assigned differently depending on the state of training of the agent. For example, considering only the short-term consequences of the actions at the beginning of training can enable the agent to get off the ground and start exploring the environment more effectively. As learning proceeds and the agent masters the basic dynamics of the environment, it may be beneficial to extend the credit assignment horizon to account for the rewards far in the future. We enable this adaptive credit assignment by parameterizing the credit assignment function with meta-parameters that are tuned by another learning algorithm online. This constitutes a bilevel optimization setting, where the inner learning algorithm, also known as the inner-loop, is a policy gradient algorithm using the parameterized credit assignment function. To achieve the goal of maximizing the returns collected by the policy, the outer algorithm optimizes the meta-parameters to improve the returns as much as possible in every iteration of the inner-loop. We do this by computing the gradient of the return of a policy updated by the inner-loop with respect to the meta-parameters. Because the outer-loop computes another gradient of a return, we use what is effectively a policy gradient method as the outer-loop algorithm as well. The gradient of the return of the updated policy w.r.t. the meta-parameters is often referred to as the *meta-gradient*. We show that using meta-gradients to learn the adaptive heuristic can significantly improve credit assignment over commonly used heuristics in difficult credit assignment scenarios with delayed rewards and that it can be used with low overhead in high-dimensional environments.

In addition to learning credit assignment heuristics, meta-gradients can be used to learn many other kinds of meta-parameters, such as the initial parameters of the policy in multi-task settings [Finn et al., 2017a] or even inner-loop parameter updates fully represented as neural networks [Oh et al., 2020]. In chapter 3, we present a survey of meta-RL methods using this approach. We discuss the

different problem settings meta-gradients have been studied in, cover the various meta-parameterizations, and discuss challenges in estimating the meta-gradients.

In chapter 5, we take a deeper look at the estimation of meta-gradients. When learning the meta-parameters online during the learning of the policy, we need to decide how often we update the meta-parameters. Updating the meta-parameters often results in a myopic approximation of the meta-gradient, which ignores the long-term consequences of the inner-loop parameter updates. Alternatively, we may choose to update the meta-parameters less often, resulting in a more difficult gradient approximation problem but accounting for the long-term effect of the updates. An additional challenge in the estimation of the meta-gradients is that the data we use for computing them depend on the policy that is being updated by the meta-gradients. Correct accounting of these dependencies has been a popular subject of research and has caused some confusion in the field [Al-Shedivat et al., 2017, Fallah et al., 2021, Tang, 2022]. Choosing how to account for the effect of all of the policies computed by the inner-loop during the optimization results in a bias-variance tradeoff in the meta-gradient estimation. We provide theoretical results dispelling some of the confusion around the correct estimation of the meta-gradients and provide an empirical study on the bias-variance tradeoff.

1.2 Imitation Learning

IL is a closely related topic to RL that also studies interactive agents. Instead of maximizing rewards, IL agents are trained to mimic behaviors demonstrated by some experts. To extend the driving example above, consider recordings of professional human drivers making their way through the scenario efficiently and safely. An IL algorithm would take these demonstrations and fit a machine learning model to maximize the probability that it takes the same actions in the same states as the humans did. Similar algorithms that resulted in the remarkable advances in AI by fitting data distributions discussed above could be employed here. Compared to the rewards in RL, the expert demonstrations contain a much less noisy signal of the desired driving behavior. Therefore, IL algorithms can learn effective policies

using fewer training resources than their RL counterparts. However, fitting a static data distribution using an ML model is an inherently limited approach to solving interactive problems and results in different kinds of failure modes of IL [Ortega et al., 2021, Ross et al., 2011, Weihs et al., 2021]. In this thesis, we look at particular kinds of failure in which the experts providing the demonstrations observe the world differently than the imitators. This results in situations where the imitator cannot know the information that lead the expert to choose its actions and can result in random behavior. This problem is known as the *imitation gap*.

This difference in the observations can arise in the case where the expert has superior sensors to view the world through compared to the imitator. For example, think of a human driver providing demonstrations to learn a driving policy. The driving policy may be limited by sensors that do not have the same richness as human senses. As a result, the expert may base its action choices on information that is not available to the imitator. In chapter 7, we consider this additional information available to the expert as a *confounder* in a causal graph [Pearl, 2009]. In this case, in order to successfully imitate the expert, the imitator has to consider the information it gathers throughout its interaction with the environment. Naively maximizing the likelihood of expert actions in this model results in causal delusions [Ortega et al., 2021], which can cause the imitator to fail. We propose to circumvent this problem by learning a separate inference model for the confounding information and condition the imitator on the inference. We show that this approach converges to the optimal imitator in theory under strong assumptions. Furthermore, we develop a practical method for learning the inference model and imitator in conjunction and show that it is able to learn successful imitators in complex environments where competing approaches fail.

This problem becomes even more difficult if the environment cannot be safely explored by trying the different policies corresponding to the inferred latent. For example, if the driving policy acts on the mistaken inference that the road is free of ice when, in fact, it is not, the policy may drive too fast and cause a collision. In such a case, some other source of information is required to learn safe behavior for

exploring the environment in addition to the expert demonstrations. In chapter 8, we propose a Bayesian approach to the imitation gap problem, where we model the extra information as a Bayesian prior over the possible behaviors. We show that in theory, our method converges to a policy that is optimal in a Bayesian sense of incorporating the information from the prior and demonstrations into its behavior as well as possible. Based on these theoretical insights, we develop a practical algorithm that learns the Bayes-optimal policy. We demonstrate the effectiveness of the algorithm in multiple imitation gap problems.

1.3 Contributions

The contributions of this thesis are divided into two parts. In the first part, we tackle the meta-RL problem, where we use meta-learning to develop more sample efficient RL algorithms. In the second part, we look at the imitation gap problem in IL. We present algorithms that enable learning the desired imitators when the expert observes the world differently from the imitator, i.e., imitation learning in the imitation gap setting. Next, we present the structure of the thesis discuss the original contributions it includes.

This thesis is divided into nine chapters as follows.

- **Chapter 1** is this chapter, in which we motivate the work conducted in this thesis, introduce the meta-RL and imitation gap problems on which this thesis focuses, and provide a high-level summary of the work conducted as part of this thesis.

- **Chapter 2** presents the mathematical definitions of the settings of the RL, meta-RL, and IL problems and describes the standard concepts and algorithms on which the contributions of this thesis are based.

Part I consists of three chapters that cover the contributions this thesis makes to meta-RL.

- **Chapter 3** provides a deeper look at meta-gradient learning for meta-RL in the form of a literature review. In the review, we survey different meta-gradient estimators, and meta-parameterizations optimized by those algorithms. This review is based on the meta-RL survey paper [Beck et al., 2023].

- **Chapter 4** presents the first algorithmic contribution of this thesis. In it, we describe a meta-parameterization for credit assignment in RL and a meta-gradient algorithm for adapting the meta-parameters online. This chapter is based on a paper published at AAAI Conference [Zheng et al., 2021].

- **Chapter 5** takes a deeper look at the estimation of meta-gradients. In it, we provide an analysis dispelling some misunderstandings about the bias-variance tradeoff in meta-gradients and conduct an empirical study to identify the Pareto frontier of the tradeoff. This chapter is based on a paper published at the NeurIPS Deep RL Workshop [Vuorio et al., 2021].

Part II consists of three chapters that present the contributions this thesis makes to solving the imitation gap problem.

- **Chapter 6** is a literature survey on imitation learning and the imitation gap problem.

- **Chapter 7** presents an algorithm to learn imitators in an imitation gap problem setting, where the extra information available to the expert is modeled as a latent confounder in a causal graph. This chapter is based on a paper published in TMLR [Vuorio et al., 2024b].

- **Chapter 8** presents an algorithm for a slightly different imitation gap setting where another source of information is required in addition to expert demonstrations. We model this information as a Bayesian prior. This chapter is based on a paper published as an arXiv preprint [Vuorio et al., 2024a].

Finally, we conclude the thesis in **chapter 9** where we discuss the contributions made in this thesis and point out promising directions for future work.

2
Background

Contents

In this chapter, we introduce the mathematical and algorithmic background of this thesis. This includes a description of the central mathematical models of Markov decision processes (MDP) and the relevant variants of contextual MDPs (CMDPs) and partially observable MDPs (POMDP). In addition to the problem settings, we describe some of the central RL, meta-RL, and IL algorithms that the work presented in this thesis is based on. The treatment of the mathematical concepts and algorithms in this chapter is necessarily somewhat superficial, but we provide pointers to original works and other materials for the interested reader. We provide any necessary chapter-specific background in the respective chapters.

2.1 Reinforcement Learning

The RL problem is learning an agent that maximizes the expected value of the rewards it gets when it interacts with an environment. RL is a popular framework for studying real-world agents controlling physical systems such as robots as well as virtual ones such as language models. We use the RL framework and concepts from RL throughout the thesis.

2.1.1 Problem Setting

Markov Decision Process The RL problem setting commonly considers a Markov decision process [Bellman, 1957, Sutton and Barto, 2018, MDP] as the model for the environment. The eponymous Markov property helps make the MDP a practical setting for developing RL algorithms. This means that the *state* of the environment depends only on the previous state and none of the states preceding it. This greatly reduces the number of dependencies that need to be considered to optimally select an action, which makes the problem more tractable in theory and in practice. Formally, an MDP is defined as a tuple

$$\mathcal{M} \coloneqq \langle \mathcal{S}, \mathcal{A}, p(s_{t+1}|s_t, a_t), p(s_0), r(s_t, a_t), \gamma \rangle, \tag{2.1}$$

where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces respectively, $p(s_{t+1}|s_t, a_t)$ the transition dynamics, $p(s_0)$ the initial state distribution, $r(s_t, a_t)$ the reward function, and $\gamma \in [0, 1)$ a discount factor. We denote a sampled reward r. The agent is modeled as a policy $\pi(a|s)$, which defines the probability of actions in a state. In the remainder of this chapter, we consider discrete distributions for the MDP and policy, but continuous variants are easily found by substituting the appropriate densities. In the empirical chapters, we consider both discrete and continuous MDPs and policies.

The agent starts its interaction with the environment in an initial state s_0 sampled from the initial state distribution $p(s_0)$. Given a state s_t, the policy defines the distribution over action probabilities $\pi(a_t|s_t)$. An action a_t is sampled from the policy and fed to the environment. Given the previous state and an action, the next state is sampled from the dynamics distribution $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. The

agent receives the next state s_{t+1}. A corresponding reward r_{t+1} is computed by the reward function. Depending on the specifics of the environment, the reward function may be deterministic or stochastic and it may depend on the full transition, e.g., $r_{t+1} = r(s_t, a_t, s_{t+1})$, or only on some elements of it. However, an important attribute of a reward function in an MDP is that it is Markovian, i.e., the reward only depends on the current state (or transition) and not on the path of how the agent got there.

Contextual MDPs Besides regular MDPs we look at learning policies in a setting where the agent faces a different initially unknown MDP at every episode. We model this problem setting as a contextual MDP (CMDP) [Hallak et al., 2015]. Formally, we use a definition for a CMDP closely related to the MDP definition given by equation (2.1):

$$\mathcal{M}(\theta) \coloneqq \langle \mathcal{S}, \mathcal{A}, p(s_{t+1}|s_t, a_t, \theta), p(s_0), r(s_t, a_t), \gamma \rangle, \tag{2.2}$$

where the dynamics distribution is parameterized by a context θ with an underlying distribution $p(\theta)$. In our setting, we assume that all the tasks in the CMDP have a common goal and therefore the reward function is independent of θ. This can model for example situations like driving in an intersection where the goal is to always cross the intersection quickly but safely but the traction of the road surface and therefore the safe maximum speed varies. We give the remaining definitions in this chapter assuming a non-contextual MDP but discuss their context-dependent variants later as appropriate.

Partially Observable MDPs Another common environment variation in RL research is the Partially Observable Markov Decision Process (POMDP) [Åström, 1965, Kaelbling et al., 1998]. POMDPs are characterized by the tuple

$$\mathcal{M} \coloneqq \langle \mathcal{S}, \mathcal{A}, p(s_{t+1}|s_t, a_t), r(s_t, a_t), \Omega, O(s_t), \gamma \rangle, \tag{2.3}$$

where Ω is the observation space and $O : \mathcal{S} \to \mathcal{P}(\Omega)$ is an observation function. In a POMDP, the agent observes the environment through the observation function O, which maps states to probability distributions over the observation space Ω.

Partial observability may result, for example, from the limited field of view of the sensors in a self-driving car. To safely navigate through an intersection, a driving agent with a restricted field of view may need to drive more cautiously than an agent with full observability, allowing extra time to stop the car if an initially obscured vehicle with the right of way becomes visible.

Trajectories and Episodes An agent's interaction with the environment defines a state-action trajectory $\tau_i \coloneqq \{s_0, a_0, s_1, a_1, \dots\}$. Depending on the specific MDP considered, the duration of the interaction of the agent with the environment may be fixed or variable and it may be finite or infinite. The probability of a trajectory is given by

$$p_{\pi,\mathcal{M}}(\tau) = p(s_0) \prod_{t=0}^{\infty} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t). \tag{2.4}$$

It is sometimes convenient to include terminal states, which mark the end of meaningful interaction between the agent and the environment and as a result the accumulation of rewards stops. In the variable length case, entering these states terminates the trajectory. In the fixed length case, a *sink state* may be introduced, which the agent enters from the terminal state and which only transitions back to itself regardless of the action. The reward function typically returns zero for any action in the sink state, which terminates the agent's need to consider future rewards even if the trajectories continue infinitely. A trajectory starting from an initial state and terminating in a terminal state or after the fixed number of steps T has passed is often termed an *episode*.

RL Objective We focus on policies with parameters $\eta \in \mathbb{R}^{d_\eta}$. The goal in reinforcement learning is to optimize the policy, or its parameters, to maximize the expected discounted return

$$J(\eta) = \mathbb{E}_{\pi_\eta,\mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{2.5}$$

where the expectation is with respect to the distribution of infinite length trajectories defined by the MDP $\mathcal{M}$ and the policy defined by the policy π_η as given

by equation (2.4). The discount factor γ is required since the infinite sum of rewards may not converge. The discount factor determines the effective horizon of future rewards the agent needs to consider when choosing actions. It can be thought of as representing the uncertainty about whether the agent has to be reset after a timestep t, e.g., a robot falls over and is unable to continue its task [Sutton and Barto, 2018]. Because changing the discount factor can change the optimal policy for the MDP, we include the discount factor in the definition of the MDP. For brevity, the return on the trajectory τ is written as $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$, where t indexes the states and actions on τ.

2.1.2 RL Methods

Value Functions Developing RL algorithms requires being able to compare the expected returns achieved by different policies in different states and after taking different actions. Value functions are the central abstraction in RL for this purpose [Sutton and Barto, 2018]. A state-value function V^π is defined as

$$V^\pi(s) = \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s \right], \tag{2.6}$$

where the first state from which the return is computed is fixed as s and the policy π is followed afterwards. A corresponding action-value function Q^π is defined as

$$Q^\pi(s, a) = \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a \right], \tag{2.7}$$

where both the initial state s_0 and the first action following it a_0 are fixed to the inputs and the policy π is followed after the first transition. Another useful expectation is captured by the advantage function defined as

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s), \tag{2.8}$$

which measures how much better in expectation it is to take the action a in state s than following the policy π from the state. For brevity we omit the superscript π on V, Q, and A, whenever the policy defining the value functions is clear from the context.

Policy Gradients A big category of algorithms for maximizing the RL objective given by equation (2.5) is centered around directly estimating the gradient of the objective with respect to the policy parameters. This gradient is known as the *policy gradient* and can be stated as

$$\nabla_\eta J(\eta) \propto \mathbb{E}_{\pi_\eta, \mathcal{M}} \left[Q(s,a) \nabla_\eta \log \pi_\eta(a|s) \right]. \tag{2.9}$$

This simple expression belies a nontrivial result known as the *policy gradient theorem* [Marbach and Tsitsiklis, 2001, Sutton et al., 2000], which proves that the policy gradient is an unbiased estimator of the gradient of the performance of the policy in an MDP. The nontrivial part is showing that the policy gradient correctly accounts for the effect the policy has on the state visitation distribution. For a detailed derivation on the policy gradient theorem we recommend Weng [2018] and Sutton and Barto [2018].

Typically, the expectation in equation (2.9) cannot be evaluated exactly and is instead approximated from samples by

$$\nabla_\eta J(\eta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{s,a \in \mathcal{D}} \nabla_\eta \log \pi_\eta(a|s) Q(s,a), \tag{2.10}$$

where $\mathcal{D}$ is a batch of N transitions (s, a, r) collected with the policy π_η. To compute this stochastic update in practice, the action-value $Q(s,a)$ needs to be estimated via, e.g., a truncated sample of the return or a suitable function approximator.

We sometimes overload the notation for the policy and write $\pi(\tau)$ to denote the product of action probabilities defined by the policy over the trajectory. In this case, we use an alternative expression for the policy gradient for convenience: $\mathbb{E}_{\pi_\eta} \left[R(\tau) \nabla_\eta \log \pi_\eta(\tau) \right]$. This remains unbiased but technically has higher variance than the one given by equation (2.10) due to the terms in the product multiplying the full return along the trajectory. In practice, due to the causal structure of the MDP, where the next state s_{t+1} only depends on the current state s_t and action a_t, the extra terms can be safely ignored without introducing bias [Weng, 2018].

Baseline and Advantage Functions The variance of the policy gradient can be reduced by subtracting a baseline function $b(s)$ from the action-value in the policy gradient

$$\nabla_\eta J(\eta) \propto \mathbb{E}_{\pi_\eta, \mathcal{M}} \left[(Q(s,a) - b(s)) \nabla_\eta \log \pi_\eta(a|s) \right]. \tag{2.11}$$

The baseline does not introduce bias to the policy gradient as long as it does not depend on the action taken by the policy or otherwise change the expectation. A convenient choice for a baseline, which does not add bias, is the state-value function $V(s)$. Using $V(s)$ as the baseline makes the multiplier for the gradient term in the policy gradient expression equivalent to the advantage function $A(s,a)$. The policy gradient with advantage is then

$$\nabla_\eta J(\eta) \propto \mathbb{E}_{\pi_\eta, \mathcal{M}} \left[A(s,a) \nabla_\eta \log \pi_\eta(a|s) \right]. \tag{2.12}$$

For a discussion on baselines and the variance of policy gradient estimators, see Greensmith et al. [2004].

In practice, the true value function V is usually not known. Instead, a Monte-Carlo approximation v may be used instead. This leads to a MC estimation of A:

$$\hat{A}^{\text{MC}}(s_0, a_0) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) - v(s_0). \tag{2.13}$$

However, this advantage estimator generally suffers from high variance. To reduce the variance, an eligibility trace parameter λ introduced by the TD(λ) algorithm [Sutton, 1988] may be used. This results in a new return estimator, called λ-return, which is a weighted sum of the n-step truncated returns, where the bias from truncation is corrected by adding the estimate of the value in the state after n-steps. The corresponding λ-estimator for advantage is

$$\hat{A}^{(\lambda)}(s_0, a_0) = \sum_{t=0}^{\infty} (\gamma\lambda)^t \delta(s_t, a_t), \tag{2.14}$$

where $\delta(s_t, a_t) = r(s_t, a_t) + \gamma v(s_t) - v(s_{t-1})$ is the TD-error at time t. This estimator recovers the MC estimator when $\lambda = 1$. For a full derivation of the λ-estimator and other advantage estimators see Schulman et al. [2015].

2.2 Meta-Reinforcement Learning

RL algorithms, such as the policy gradient methods introduced above, are notoriously sample inefficient. In this thesis, we look into learning more sample efficient learning algorithms as one idea for reducing this inefficiency. We adopt a meta-RL setting, where an RL algorithm has some *meta-parameters* that can be learned via gradient descent on a meta-learning objective. More specifically we consider the widely studied case where both the RL algorithm being learned and the meta-learning algorithm are policy gradient algorithms [Al-Shedivat et al., 2017, Fallah et al., 2020, Finn et al., 2017a]. We often refer to the inner RL algorithm as the *inner-loop* and the outer RL algorithm correspondingly as the *outer-loop*.

Update Function The inner-loop defines an *update function* U, which computes updates to the parameters of a policy and is itself parameterized by the meta-parameters ϕ. We consider update functions of the form

$$U_\phi(\eta, \mathcal{D}) : \mathbb{R}^{d_\eta} \times \{(\mathcal{S}, \mathcal{A}, \mathbb{R})\}^N \to \mathbb{R}^{d_\eta}, \tag{2.15}$$

where $\mathcal{D}$ is a batch N transitions (s, a, r). Sometimes it is convenient to consider batches consisting of whole trajectories τ. For example, a simple update function is $\eta^{i+1} = \eta^i + U_\phi(\eta^i, \mathcal{D}^i) = \eta^i + \phi \nabla_{\eta^i} J(\eta^i, \mathcal{D}^i)$, where the meta-parameter $\phi \in \mathbb{R}$ is the learning rate of a stochastic policy gradient update, and i indexes the sequence parameters starting from the initial policy parameters and updated using U_ϕ. In general, the meta-parameter ϕ can have an arbitrary number of dimensions.

Meta-RL Problem Setting The meta-RL problem setting considers a distribution of tasks $p(\mathcal{M})$, which are modeled as MDPs [Beck et al., 2023]. In the beginning of each task, the agent does not know which task it is in and it has to explore the environment and adapt its policy according to the rewards it receives. Therefore, a single Markovian policy, like those learned using policy gradient methods described above, is not enough to solve a general meta-RL problem. Instead, the inner-loop represents a whole RL algorithm, which controls how to change the policy

in response to the data it receives. The performance of a meta-RL algorithm is measured by the returns it receives in expectation across the whole task distribution. The meta-RL objective can be stated as

$$\mathcal{J}(\phi) = \mathbb{E}_{\mathcal{M}^i \sim p(\mathcal{M})}\left[\mathbb{E}_{\mathcal{D}}\left[\sum_{\tau \in \mathcal{D}_{K:H}} R(\tau) \middle| U_\phi, \mathcal{M}^i\right]\right], \tag{2.16}$$

where the inner expectation is taken over data sampled with all of the policies along the update trajectory so far, as each η^k depends on all of the previous data through U_ϕ, the index K represents the index of the first trajectory collected by the inner-loop that counts toward the meta-RL objective, and H determines how long the inner-loop interacts with the environment for. K and H are referred to as the *shot* and *task-horizon* or *lifetime* of the meta-RL problem respectively. The shot K is introduced, because sometimes it is useful to allow the inner-loop some free exploration in the beginning of the interaction with the new task before its performance starts counting toward the outer-loop objective. The trajectories τ are sampled from the MDP $\mathcal{M}^i$ with the policy π_η, whose parameters are produced by the inner-loop update function U_ϕ.

2.2.1 Meta-Gradients

The gradient of the meta-RL objective given by equation (2.16) w.r.t. the meta-parameters is often known as the *meta-gradient*. Estimating meta-gradients has been an active area of research, which we survey in chapter 3 and analyse more deeply in chapter 5. Here, we present an unbiased meta-gradient estimator originally derived by Al-Shedivat et al. [2017].

The Hth parameter in a sequence of parameter updates produced by the update function U_ϕ is

$$\eta^H = \eta^0 + \sum_{i=0}^{H-1} U_\phi(\eta^i, \mathcal{D}^i), \tag{2.17}$$

where the $\mathcal{D}^i$ are random variables representing the batches the updates are computed on. The probability of the batch is $p_\eta(\mathcal{D}) = \prod_{\tau \in \mathcal{D}} p_\eta(\tau)$. The policy parameters η^i for $i > 0$ depend on the initial parameters η^0, the meta-parameter

ϕ, and the data and are therefore random variables too. Since the meta-gradient estimation does not interact with the task selection we ignore the task distribution in the following. In practice, the task distribution is simple to account for by taking an expectation over it when meta-learning in a multi-task setting. An unbiased meta-gradient estimator for the objective in equation 2.16 with $K = 0$, on the sequence of updates in equation 2.17 can be stated as follows:

$$\begin{aligned}\nabla_\phi J_H(\phi) &= \nabla_\phi \sum_{h=0}^{H} \mathbb{E}_{\{\mathcal{D}^i\}_{i=0}^{h-1}} \Big[\mathbb{E}_{\tau \sim p_{\eta^h}(\tau)} \left[R(\tau) \right] \Big] \\ &= \sum_{h=0}^{H} \mathbb{E}_{\substack{\{\mathcal{D}^i\}_{i=0}^{h-1} \\ \tau \sim p_{\eta^h}(\tau)}} \Bigg[\Bigg(\underbrace{\sum_{j=0}^{h-1} \nabla_\phi \eta^j \nabla_{\eta^j} \log p_{\eta^j}(\mathcal{D}^j)}_{\texttt{sampling correction}} + \underbrace{\nabla_\phi \eta^h \nabla_{\eta^h} \log p_{\eta^h}(\tau)}_{\texttt{direct meta-gradient}} \Bigg) R(\tau) \Bigg], \end{aligned} \tag{2.18}$$

where $\nabla_\phi \eta^h \in \mathbb{R}^{d_\phi \times d_\eta}$ is the derivative w.r.t. ϕ of the sequence of updates resulting in η^h. The terms of the sum $\sum_{j=0}^{h-1} \nabla_\phi \eta^j \nabla_{\eta^j} \log p_{\eta^j}(\mathcal{D}^j)$ give the *sampling correction* that assigns credit from the experience collected with the updated policy directly to the earlier policies. We separate the sampling correction and direct meta-gradient terms because the former is often omitted from practical meta-gradient estimators. The bias-variance tradeoff resulting from the omission is further discussed in chapter 5. Following from the additive sequence of updates in equation 2.17, we can expand $\nabla_\phi \eta^h$ as

$$\nabla_\phi \eta^h = \nabla_\phi \eta^0 + \sum_{i=0}^{h-1} \Big(\nabla_\phi U_\phi(\eta^i, \mathcal{D}^i) + \nabla_\phi \eta^i \nabla_{\eta^i} U_\phi(\eta^i, \mathcal{D}^i) \Big). \tag{2.19}$$

An influential meta-gradient algorithm, model-agnostic meta-learning (MAML) [Finn et al., 2017a], considers the initial parameters of the policy the meta-parameters: $\phi \coloneqq \eta^0$. Therefore, in MAML and related algorithms, $\nabla_\phi \eta^0$ can be nonzero. The terms of the form $\nabla_\phi \eta^h \nabla_{\eta^h} \log p_{\eta^h}(\tau)$ are the product of the derivative of the hth policy parameter w.r.t. the meta-parameters and the standard policy gradient w.r.t. η^h.

A common heuristic used by meta-gradient methods is to apply a policy gradient algorithm directly on the returns $\sum_{k=K}^{H} R(\tau_k)$ [Finn et al., 2017a, Xu et al., 2020]. However, this does not compute exactly the same meta-gradient as derived above because it does not include the sampling correction terms. As found by [Fallah et al.,

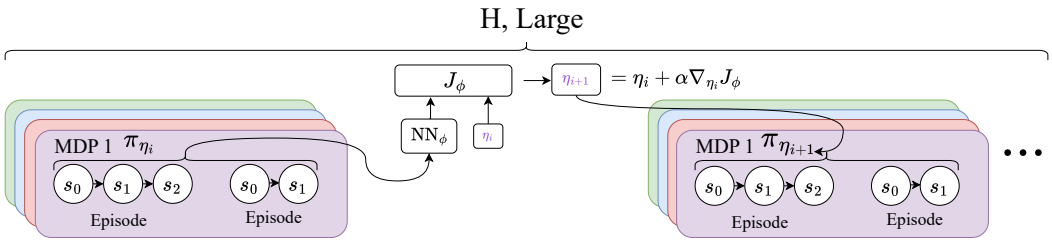


Figure 2.1: Often in many-shot meta-RL, the meta-parameters ϕ parametrize a policy gradient objective function J_ϕ, which is used for updating the parameters of the policy η. This incorporates the inductive bias of a policy gradient algorithm into the inner-loop, which helps improve policies over long trials. The outer-loop updates the meta-parameters after a number of inner-loop updates. The number of inner-loop updates between each outer-loop update depends on the specifics of the problem setting and method.

2020, Stadie et al., 2018] and verified in chapter 5, the bias resulting from omitting the sampling correction terms can sometimes be detrimental to the meta-learning performance but the high variance of the unbiased estimator often dominates.

Evolution Strategies for Meta-Gradient Estimation A popular alternative for computing the meta-gradient via backpropagation is to use a black box optimization method called Evolution strategies (ES) [Rechenberg, 1971, Schwefel, 1977]. ES for meta-RL works by sampling a set of candidates for the meta-parameters ϕ from a parameterized distribution: $p_\psi(\phi)$. Then, ES approximates gradient ascent using a score function estimator [Williams, 1992] of the meta-RL objective given by equation (2.16). This corresponds to optimizing a smoothed version of the true objective. Using ES can result in a lower variance meta-gradient estimate, because the objective surface for meta-RL with many inner-loop updates may not be smooth [Metz et al., 2019]. However, ES has the drawback that its sample complexity scales linearly with the dimension of the optimized parameters [Nesterov and Spokoiny, 2017].

2.2.2 Many-Shot Meta-RL

Lots of research has been conducted on the few-shot meta-RL setting, where the goal is to produce a policy for a new task after a handful of episodes. In contrast, in part I,

we focus instead on a many-shot meta-RL setting, where the inner-loop may interact with the task for tens of thousands or more episodes. In many-shot meta-RL, the meta-parameters ϕ typically parametrize a policy gradient objective function J_ϕ, which is used to update the policy parameters η in the inner-loop. The goal is to improve the policy over long optimization runs, sometimes referred to as trials, where the meta-parameters are updated after a series of inner-loop updates. This setup contrasts with few-shot meta-RL, where adaptive policies learn to solve unseen tasks in only a few episodes by leveraging task distribution knowledge. For more complex task distributions, few-shot approaches may struggle, requiring many-shot methods that resemble standard RL algorithms to improve sample efficiency. Multi-task many-shot meta-RL aims to learn algorithms capable of quickly adapting to new tasks across a broader range of task distributions, optimizing meta-parameters after multiple inner-loop updates. For example, Oh et al. [2020] use a many-shot meta-RL algorithm to learn an objective function in the inner-loop, which generalizes from a set of simple MDPs to computer games on the Atari console.

Learning inner-loops that work over many updates is computationally demanding, because backpropagation through the sequence of updates usuallly requires storing the entire inner-loop in memory. Furthermore, meta-gradients over long sequences of updates may have high variance. Due to these limitation, surrogate objectives based on truncated trials are often used to update meta-parameters, which introduces bias but remains practical. The pseudocode for many-shot meta-RL algorithm 1 illustrates the iterative process of inner-loop updates followed by meta-parameter updates. To provide further intuition, we present the algorithm visually in figure 2.1.

Single-Task Meta-RL Besides solving more complex task distributions, many-shot meta-RL is useful as a tool for accelerating regular single-task RL. We refer to this setting as single-task many-shot meta-RL. In this setting, meta-learning improves sample efficiency during training by updating the meta-parameters while the policy continuously evolves throughout the agent's lifetime. Unlike multi-task methods, which re-initialize policy parameters for each task, single-task approaches

Algorithm 1 Many-Shot Meta-Learning for Reinforcement Learning

Initialize meta-parameters ϕ
Sample a batch of tasks $\mathcal{M}^i \sim p(\mathcal{M})$
Initialize policy parameters η_0^i for each task $\mathcal{M}^i$
Set $j = 0$
while not done **do**
    for N iterations **do**
        for each task $\mathcal{M}^i$ **do**
            Collect data $\mathcal{D}_j^i$ using the policy $\pi_{\eta_j^i}$
            Update policy parameters: $\eta_{j+1}^i = \eta_j^i + \alpha \nabla_{\eta_j^i} J_\phi(\mathcal{D}_j^i, \pi_{\eta_j^i})$
        end for
        $j = j + 1$
    end for
    Update meta-parameters: $\phi \leftarrow \phi + \beta \nabla_\phi \sum_i J(\mathcal{D}_j^i, \pi_{\eta_j^i})$
    For tasks i for which the trial has concluded, re-initialize the policy parameters η^i, and sample new tasks $\mathcal{M}^i \sim p(\mathcal{M})$.
end while

maintain a single set of parameters. As the policy changes, the non-stationary nature of the training environment presents a unique challenge. The algorithm template given by algorithm 1 can be applied in the single-task case by choosing a task distribution with only one task and choosing the N such that the inner-loop is updated regularly. We study single-task meta-RL methods in part I.

2.3 Imitation Learning

So far, we have focused on RL, where an agent interacts with the environment and learns to maximize the expected rewards it collects during its interaction with the environment. Besides the sample inefficiency issue discussed above, RL can be challenging to apply due to the difficulty of defining good reward functions. Luckily, when demonstrations of the desired behavior are available, learning policies directly from the demonstrations may be possible and some approaches are computationally more efficient than RL. Learning from demonstrations is alternatively known as imitation learning. Next, we consider different kinds of imitation learning approaches relevant to this thesis. In section 2.3.2, we discuss the imitation gap problems that are the focus of part II of this thesis.

2.3.1 IL Methods

Behavioral Cloning Arguably the simplest form of imitation learning is known as behavioral cloning (BC) [Pomerleau, 1988]. In BC, an expert provides a dataset of demonstrations and a policy is learned on that dataset to maximize the likelihood of the expert actions via supervised learning. The expert is a policy that acts in an MDP. The expert may maximize the expectation over some reward function, but this is not necessary (and some tasks cannot be expressed through Markov rewards [Abel et al., 2021]). A BC policy $\pi_\eta(a \mid s)$ parametrized by η is learned by maximizing the likelihood of the expert actions, i.e., minimizing the loss

$$\mathcal{L}_{\mathrm{BC}}(\eta) = -\mathbb{E}_{\pi_{\mathrm{exp}},\mathcal{M}} \left[\log \pi_\eta(a \mid s)\right], \tag{2.20}$$

where the expectation is over the data distribution defined by the interaction of the expert's policy π_{exp} with the MDP $\mathcal{M}$. The expectation is often estimated from a fixed dataset of sampled expert trajectories. We often refer to the policy imitating the expert as the *imitator*. At deployment time, the imitator can be used for choosing actions in the same MDP.

BC is often a computationally efficient way of learning a policy assuming high quality demonstrations are available. However, a common issue with BC is that during training, the imitator only sees the states the expert visits and learns to prefer the actions the expert chose in those states. At deployment time, the imitator chooses the actions and as a result may end up in states the expert never visited. This problem is compounded over longer trajectories as the earlier errors in action selection result in further deviations from the expert trajectories. This is known as the *distribution shift* issue in BC and it can make collecting sufficient expert data infeasible.

Dataset Aggregation (DAgger) A popular mitigation to the distribution shift issue discussed above is to use the imitator to collect data from the MDP during the imitation learning and using the expert policy to label those data with the correct actions. This is known as dataset aggregation [Ross et al., 2011]. DAgger

can enable imitation learning in settings where collecting sufficient BC datasets is infeasible. However, the requirement for having *query access* to the expert policy, i.e., the ability to use the expert policy to choose actions in arbitrary states during data collection, can be prohibitive. For example, if the expert is a human, applying DAgger would require a human labeler to be available throughout the BC training.

Inverse RL Inverse RL (IRL) is a problem setting related to imitation learning, where instead of learning a policy that matches the expert behavior directly, the goal is to find a reward function that explains the expert behavior as well as possible. As the name suggests, this can be seen as an inversion of the RL problem where the goal is to learn a policy that maximizes a reward. The IRL problem can be expressed as finding the parameters $\omega \in \mathbb{R}^{d_\omega}$ of a reward function r_ω such that

$$\mathbb{E}_{\pi_{\text{exp}},\mathcal{M}}\left[\sum_{t=0}^{\infty}\gamma^t r_\omega(s_t,a_t)\right] \geq \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t=0}^{\infty}\gamma^t r_\omega(s_t,a_t)\right] \qquad \forall\pi, \tag{2.21}$$

where π_{exp} is the expert policy. However, this formulation is not enough for identifying reward functions that capture the expert intent unambiguously because it allows for degenerate reward functions such as the one that is zero everywhere and furthermore it allows infinitely many solutions [Ng et al., 2000].

To find unambiguous and nondegenerate reward functions, a principled approach is to choose one, which follows the maximum entropy principle [Jaynes, 1957] in making the fewest number of assumptions besides matching the distribution generated by the expert policy [Ziebart et al., 2008a]. Consider a reward function $r_\omega(s,a) = \nu(s,a)^\top\omega$, where $\nu \in \mathbb{R}^{d_\nu}$ is a feature representation of the state-action, and ω are the reward function parameters. Then, Ziebart et al. [2008a] propose to learn a policy that is as random as possible while matching the expert features

$$\begin{aligned}&\max_{\omega}\mathcal{H}(\pi_{r_\omega})\\ &\text{such that } \mathbb{E}_{\pi_{r_\omega},\mathcal{M}}\left[\sum_{t=0}^{T}\nu(s_t,a_t)\right] = \mathbb{E}_{\pi_{\text{exp}},\mathcal{M}}\left[\sum_{t=0}^{T}\nu(s_t,a_t)\right],\end{aligned} \tag{2.22}$$

where π^{r_ω} is the optimal maximum entropy policy. The optimal policy for the maximum entropy problem is sometimes called the soft-optimal policy. The

maximum entropy IRL problem is equivalent to maximizing the likelihood of the expert data under the model

$$p(\tau|\omega, \mathcal{M}) \approx \frac{\exp(\sum_{t=0}^{T} \nu(s_t, a_t)^\top \omega)}{Z(\omega, \mathcal{M})} \prod_{s_t \in \tau, a_t, s_{t+1}} p(s_{t+1}|s_t, a_t), \tag{2.23}$$

where $Z(\omega, \mathcal{M})$ is the partition function [Ziebart et al., 2008a].

The maximum entropy IRL problem defined by equation (2.22) can be modeled as a bi-level optimization problem, where the soft-optimal policy for the current reward parameters and the reward function that maximizes the likelihood of the expert data are learned in the inner-loop and outer-loop respectively. The likelihood defined by equation (2.23) can be approximately optimized by descending the gradient given by

$$\nabla_\omega \mathcal{L}(\omega) = \mathbb{E}_{\tau \sim p_{\text{exp}}} \left[\nabla_\omega \sum_{t=0}^{T} \nu(s_t, a_t)^\top \omega) \right] - \mathbb{E}_{\tau \sim p(\tau|\omega, \mathcal{M}} \left[\nabla_\omega \sum_{t=0}^{T} \nu(s_t, a_t)^\top \omega \right], \tag{2.24}$$

where the trajectory distribution $p(\tau|\omega, \mathcal{M})$ can be approximated from trajectories sampled with the soft-optimal policy [Ziebart et al., 2008a].

Generative Adversarial Imitation Learning (GAIL) Maximum entropy IRL provides a principled way to define which reward function we want to learn. However, computing the reward and the resulting policy with the bi-level optimization approach defined above is expensive because it requires solving the soft-optimal policy in the inner-loop. Viewing imitation learning as the problem of matching the expert data distribution leads to more scalable algorithms. The distribution matching problem can be expressed as minimizing the divergence between the limiting distribution over state-action pairs $D(\rho^*(s, a) \| \rho_\pi(s, a))$ where D is some statistical divergence measure and ρ^* and ρ_π are the state-action marginal distributions of the expert and the imitator policies respectively [Finn et al., 2016, Ho and Ermon, 2016].

GAIL [Ho and Ermon, 2016] is a practical algorithm for imitation learning from the distribution matching perspective. In GAIL, a discriminator network $D_\omega(\tau)$ is trained to classify the trajectories coming from the expert and the imitator. The $D_\omega(\tau)$ is then directly used as a reward. This is directly related

to generative adversarial networks [Goodfellow et al., 2014, GAN], which are an influential approach in generative machine learning for non-interactive settings such as image generation.

Successor Features For approximating the expectations required for IRL, expressing the value functions as functions of the feature expectations can be convenient. Successor features [Barreto et al., 2017, SFs] are a value function representation that decouples the dynamics of the environment from the reward. For the linear reward function $r_\omega(s,a) = \nu(s,a)^\top \omega$ where $\nu(s,a) \in \mathbb{R}^d$ are features and $\omega \in \mathbb{R}^d$ are weights. For any given policy, we may then factor the Q-function as $Q^\pi(s,a,\omega) = \Psi^\pi(s,a)^\top \omega$ where $\Psi^\pi(s,a) := \mathbb{E}_{\tau\sim p^\pi}[\sum_{t=0}^{\infty} \gamma^t \nu(s_t,a_t) | s_0 = s, a_0 = a]$ is the successor feature of (s,a) under π.

2.3.2 Imitation Gap

In part II, we present research on the imitation gap problem, where the expert and the imitator receive differing information about the environment. More specifically, we focus on the case where the expert knows something more about the world than the imitator. This can be modeled as a CMDP, where the context variable θ represents the extra information the expert observes but the imitator does not. We consider the case where only the dynamics distribution of the CMDP depends on the context but the reward function does not. This choice makes it possible to identify the correct expert policy to imitate based on observing the dynamics alone, as we propose to do in chapters 7 and 8. In these settings, just as in regular IL, the goal of the imitator is to behave like the expert. However, because the expert acts based on more information about the world than the imitator has access to, naïvely optimizing the BC objective given by equation (2.20), or any other standard IL objective for that matter, does not necessarily result in behavior that resembles the expert in the desired way [Ortega et al., 2021, Weihs et al., 2021]. Next, we discuss two kinds of problems arising from the imitation gap.

Confounded Imitation Learning In a less severe case of the imitation gap, the privileged information available to the expert precludes exactly mimicking the expert behavior but the problem has wide enough margins of error that even a naïve imitator behaves safely. This could be the case, for example, when the expert controls some physical system like the double pendulum [Spong et al., 2020] with unknown parameters but which is robust to different kinds of controllers. In such case, it may be useful to model the imitation gap problem as a problem of causal confounding in BC. The problem with using BC in this setting is that in states where the expert uses its knowledge about the latent to choose actions, the imitator marginalizes over its uncertainty about the latent variable. This results in the imitator selecting actions randomly in those states. To fix this, the imitator policy has to condition on the entire history of interaction with the environment instead of the current state alone. That is, the imitator is a non-Markovian policy defined as a function of the entire trajectory $\pi_\eta(a_t|s_0, a_0, s_1, a_1, \ldots, s_t)$. This enables the agent to learn to make an inference about the value of the latent variable based on everything it has observed so far, eventually allowing it to break ties over the actions. However, this introduces another problem where the latent variable of the environment acts as a causal confounder [Pearl, 2009] in the graph modeling the interaction of the agent with its environment. In chapter 7, we propose a method that learns a separate inference model for the latent variable θ to mitigate this problem.

Missing Exploration Information In a more severe case of the imitation gap, acting based on a guess about the privileged information results in highly undesirable behavior we want to prevent the agent from engaging in. For example, if the privileged information pertains to the location of high ledges in a locomotion environment, acting based on a mistaken guess about their location may result in a dangerous fall [Weihs et al., 2021]. In this case, even using a method such as the one presented in chapter 7 is not enough, because the naïve imitator does not result in safe exploration behavior that would enable it to make an accurate inference

about the latent variable θ. In chapter 8, we propose to deliver this information to the imitator in the form of a Bayesian prior over reward functions in an IRL setting.

Part I

Meta-Reinforcement Learning

3

Meta-RL Literature Review

Contents

3.1 Introduction

In chapter 1, we discussed how RL is a sample inefficient approach for training interactive policies and pointed to meta-RL as a potential avenue for improving the sample efficiency. The promise of meta-RL comes from using adaptive algorithms that can incorporate information they collect through their interaction with the environment to adapt their learning behavior. In chapter 2, we described the meta-RL problem setting, where an inner-loop algorithm adapts to a specific MDP sampled from a distribution of tasks and the outer-loop adapts the parameters of the inner-loop. In this chapter, we survey meta-RL literature most relevant to the original

contributions proposed in the following chapters. In particular, we discuss meta-RL algorithms, which represent the inner-loop as a policy gradient algorithm and adapt the parameters of the inner-loop using another optimization algorithm in the outer-loop. The outer-loop algorithms most commonly encountered in the literature tend to be policy gradient algorithms themselves. We divide our discussion about meta-gradient methods into two categories: few-shot meta-gradient methods discussed in section 3.2 and many-shot meta-gradient methods discussed in section 3.3.

3.2 Few-Shot Meta-Gradient Methods

In meta-RL, the inner-loop U_ϕ is learned to maximize the objective given by equation (2.16), over the task distribution, $p(\mathcal{M})$. However, during training, we generally only assume access to samples from this distribution. These samples may be limited, and the distribution over which we want to generalize may be broad. Moreover, the distribution on which the meta-RL is evaluated for meta-testing may differ from the distribution on which the meta-RL agents is meta-trained. Each of these motivates the need for learning to take place in the inner-loop algorithm U_ϕ. This section discusses methods for building such structure into the inner-loop itself.

In this section, we discuss one way of parameterizing the inner-loop that builds in the structure of existing standard RL algorithms. Meta-gradient methods are a common class of methods which parameterize the learning algorithm U_ϕ as a policy gradient algorithm. These algorithms generally have an inner-loop of the form

$$\eta_{j+1} = U_\phi(\mathcal{D}_j, \eta_j) = \eta_j + \alpha_\phi \nabla_{\eta_j} J_\phi(\mathcal{D}_j, \pi_{\eta_j}),$$

where $J_\phi(\mathcal{D}_j, \pi_{\eta_j})$ is an estimate of the returns of the policy π_{η_j}. In MAML [Finn et al., 2017a], $\phi \coloneqq \eta_0$, and so the initialization is the meta-learned component. It is also possible to add additional pre-defined components to the inner-loop, such as sparsity-inducing regularization [Lou et al., 2021]. In general, whatever structure is not predefined, is a parameter in ϕ that is meta-learned. In addition to the initialization, the meta-learned structure can include components such as hyper-parameters [Li et al., 2017] or a separate network modifying the initialization [Vuorio

et al., 2019]. Some methods also meta-learn a preconditioning matrix to approximate the curvature of the objective, inspired by second-order optimization methods. These methods generally have the form $\eta_{j+1} = \eta_j + \alpha_\phi M_\phi \nabla_{\eta_j} J_\phi(\mathcal{D}_j, \pi_{\eta_j})$ [Flennerhag et al., 2020, Park and Oliva, 2019]. While a value based-method could be used to parameterize the inner-loop instead of a policy gradient [Zou et al., 2021], value based-methods generally require many more steps to propagate reward information [Mitchell et al., 2020], and so are typically reserved for the many-shot setting, discussed in section 3.3. In this section, we focus on methods which use policy gradients in the inner-loop. We begin by discussing different parameters of the base policy that the inner-loop may adapt. Then, we discuss options for outer-loop algorithms and optimization. We conclude with a discussion of the trade-offs associated with meta-gradient methods.

3.2.1 Adapted Policy Parameters

Few-shot meta-gradient algorithms commonly learn an initialization, and then adapt that initialization in the inner-loop. Instead of adapting a single initialization, several meta-gradient methods learn a full distribution over initial policy parameters, $p(\eta_0)$ [Ghadirzadeh et al., 2021, Gupta et al., 2018, Wang et al., 2020, Yoon et al., 2018, Zou and Lu, 2020]. This distribution allows for modeling uncertainty over policies. The distribution over initial parameters can be represented with a finite number of discrete particles [Yoon et al., 2018], or with a Gaussian approximation fit via variational inference [Ghadirzadeh et al., 2021, Gupta et al., 2018]. Moreover, the distribution itself can be updated in the inner-loop, to obtain a posterior over (a subset of) model parameters [Gupta et al., 2018, Yoon et al., 2018]. The updated distribution may be useful both for modeling uncertainty [Yoon et al., 2018], and for temporally extended exploration, if policy parameters are resampled periodically [Gupta et al., 2018].

Alternatively, some meta-gradient methods adapt far fewer parameters in the inner-loop. Instead of adapting all policy parameters, they adapt a subset [Raghu et al., 2020, Zintgraf et al., 2019]. For example, one method adapts only the weights

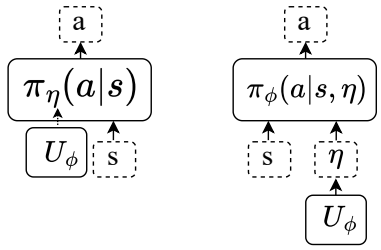


Figure 3.1: Illustration of meta-RL without (left) and with (right) a context vector. The context vector can be updated with backpropagation in a meta-gradient method or by a neural network in a black box method. When using a context vector, some of the policy parameters are not adapted by the inner-loop, and are instead meta-parameters set by the outer-loop.

and biases of the last layer of the policy [Raghu et al., 2020], while leaving the rest of the parameters constant throughout the inner-loop. Another method adapts only a vector, called the *context vector*, on which the policy is conditioned [Zintgraf et al., 2019]. In this case, the input to the policy itself parameterizes the range of possible behavior. We write the policy as $\pi_\phi(a|s,\eta)$, where η is the adapted context vector. The weights and biases of the policy, as well as the initial context vector, constitute the meta-parameters that are constant in the inner-loop. We visualize the use of a context vector in figure 3.1.

Consider the algorithm Context Adaptation via Meta-learning (CAVIA) [Zintgraf et al., 2019]. CAVIA is similar to MAML but it only adapts a context vector, which is initialized to the zero vector:

$$\eta_0 = \mathbf{0},$$
$$\eta_{j+1} = \eta_j + \alpha_\phi \nabla_{\eta_j} J_\phi(\mathcal{D}_j, \pi_\phi(\cdot|\eta_j)).$$

While the update to η is the same as in MAML, here, η represents a vector passed as an input to the policy network, while all the weights and biases of the network remain fixed throughout adaptation. One benefit of adapting a subset of parameters in the inner-loop is that it may mitigate overfitting in the inner-loop, for task distributions where only a small amount of adaption is needed [Zintgraf et al., 2019].

3.2.2 Meta-Gradient Estimation for the Outer-Loop

Effective learning in the outer-loop of meta-gradient algorithms requires access to estimates of *meta-gradients*, i.e., gradients of the outer-loop objective with respect to the meta-parameters. Most commonly, the meta-gradients are computed by directly optimizing objective given by equation (2.16) with a policy gradient algorithm. In section 2.2, we show how an unbiased meta-gradient can be derived. However, naively applying a policy gradient method in the outer-loop can lead to a suboptimal bias-variance trade-off. Significant research has considered how to improve this trade-off when estimating meta-gradients [Fallah et al., 2020, 2021, Foerster et al., 2018, Liu et al., 2022, 2019, Mao et al., 2019, Rothfuss et al., 2018, Stadie et al., 2018, Tack et al., 2022, Tang, 2022, Tang et al., 2021, Vuorio et al., 2021]. Next, we discuss the bias-variance trade-offs in meta-gradient estimation and the algorithms arising from choosing different points in the trade-off. We delve deeper into this topic in chapter 5.

Significant research has considered a meta-gradient estimator derived originally for computing higher-order meta-gradients [Farquhar et al., 2019, Foerster et al., 2018, Liu et al., 2022, 2019, Mao et al., 2019, Rothfuss et al., 2018, Tang, 2022, Tang et al., 2021]. In practice, meta-gradient algorithms use a sample-based policy gradient algorithm for updating the policy parameters in the inner-loop. The higher-order meta-gradients of the sample-based inner-loop are different from those of an inner-loop that uses the expected policy gradient. Foerster et al. [2018], Rothfuss et al. [2018] argue that the higher-order meta-gradients of the sample-based inner-loop should approximate those of the expected inner-loop and derive alternative sample-based inner-loop update functions for that purpose. While these meta-gradient estimators are still biased, their variance has a more benign dependence on the inner-loop sample size than the unbiased meta-gradient estimator and therefore can achieve a better point in the bias-variance trade-off than either the naive or the unbiased meta-gradient estimators [Liu et al., 2022, Tang et al., 2021]. To further reduce the variance of this class of meta-gradient estimators, Farquhar et al. [2019],

Liu et al. [2019], Mao et al. [2019], Rothfuss et al. [2018] propose to ignore certain high-variance terms in the estimator and introduce baselines.

Alternatively, some meta-gradient methods do not require higher-order meta-gradients. For example, Finn et al. [2017a] use a first-order approximation of the meta-gradient, whereas Song et al. [2020] use gradient-free optimization. (See Nichol et al. [2018] for another first-order approximation proposed for supervised meta-learning, and recently used in meta-RL [Ren et al., 2022].) Furthermore, when meta-learning an initialization as in MAML, for a limited number of tasks or limited amount of data in the inner-loop, it may even be preferable to set the inner-loop to take no steps at all during meta-training, i.e., without adapting to each task [Gao and Sener, 2020]. In this case, task adaptation occurs only through fine-tuning at test-time. Additionally, it is possible to use a value-based algorithm in the outer-loop, instead of a policy gradient algorithm, which avoids meta-gradients altogether [Sung et al., 2017].

3.2.3 Outer-Loop Algorithms

While most meta-gradient methods use a policy-gradient algorithm in the outer-loop, other alternatives are possible [Mendonca et al., 2019, Sung et al., 2017]. For example, one can train a critic, $Q_\phi(s, a, \mathcal{D})$, using-TD error in the outer-loop, then reuse this critic in the inner-loop [Sung et al., 2017]. Alternatively, instead of estimating meta-gradients via backpropagation, ES [Rechenberg, 1971, Salimans et al., 2017, Wierstra et al., 2014] may be used [Song et al., 2020]. Additionally, one can train task-specific experts and then use these for imitation learning in the outer-loop. While imitating experts does not lead to exploratory, they can work in some problems.

3.2.4 Meta-Gradient Trade-Offs

One of the primary benefits of meta-gradient algorithms is that they produce an inner-loop that converges to a locally optimal policy, even when relatively few samples are available for meta-training or when the task distribution differs from meta-training to meta-testing. In meta-gradient methods, the inner-loops are

typically guaranteed to converge under the same assumptions as standard policy gradient methods. For example, the MAML inner-loop converges with the same guarantees as REINFORCE [Williams, 1992], since it simply runs REINFORCE from a meta-learned initialization. Even convergence bounds are possible [Fallah et al., 2021]. However, as with any policy gradient algorithm that uses estimated gradients, the guarantees given by REINFORCE are rather weak, and in some cases a step of REINFORCE can even make the policy worse on the task [Deleu and Bengio, 2018]. Having a learning algorithm that eventually adapts to a novel task is desirable, since it reduces the dependence on seeing many relevant tasks during meta-training.

Although parameterizing U_ϕ as a policy gradient method may ensure that adaptation generalizes, this structure also presents a trade-off. Typically the inner-loop policy gradient has high variance and requires a value estimate covering the full episode, so estimating the gradient generally requires many episodes. Hence, meta-gradient methods are generally not well-suited to few-shot problems that require stable adaption at every timestep or within very few episodes in the inner-loop. Moreover, meta-gradient methods are often sample-inefficient during meta-training as well, because the outer-loop generally relies on on-policy evaluation, rather than an off-policy method that can reuse data efficiently.

In general, there is a trade-off between generalization to novel tasks and specialization over a given task distribution. How much structure is imposed by the parameterization of U_ϕ determines where each algorithm lies on this spectrum. The structure of meta-gradient methods places them near the end of the spectrum that ensures generalization. While this spectrum summarizes current methods, the trade-off is not necessarily inherent to the problem setting, and future work could investigate methods that achieve the best of both worlds. In the next section, we discuss methods at the other end of the spectrum.

3.3 Many-Shot Meta-Gradient Methods

Algorithms for many-shot meta-RL aim to improve over the plain RL algorithms they build upon by introducing meta-learned components. The choice of meta-

Meta-parameterization	Multi-task	Single-task
Intrinsic rewards	Alet et al. [2020], Meier and Mujika [2022], Veeriah et al. [2021], Zheng et al. [2020], Zou et al. [2021]	Rajendran et al. [2020], Zheng et al. [2018]
Auxiliary tasks		Flennerhag et al. [2021], Lin et al. [2019], Veeriah et al. [2019], Zahavy et al. [2020]
Hyperparameter functions		Almeida et al. [2021], Flennerhag et al. [2021], Lu et al. [2022], Luketina et al. [2022]
Credit assignment heuristics		Wang et al. [2019b], Xu et al. [2018], Zheng et al. [2021]
Hierarchies	Frans et al. [2018], Fu et al. [2020], Nam et al. [2022], Veeriah et al. [2021]	
Objective functions directly	Bechtle et al. [2021], Houthooft et al. [2018], Jackson et al. [2023], Kirsch et al. [2019], Oh et al. [2020]	Xu et al. [2020]
Optimizers	Chen et al. [2017], Lan et al. [2023]	
Black-box	Kirsch et al. [2021]	

Table 3.1: Many-shot meta-RL methods categorized by the task distribution considered and meta-parameterization.

parameterization depends on the problem. The meta-parameterizations differ in how much structure they can capture from the task distribution, which matters in the multi-task case, and what aspects of the RL problem they address. The meta-parameterization may tackle problems such as credit assignment, representation learning, etc. The best choice of meta-parameterization depends on what aspect of the problem is the primary challenge.

Many of the topics discussed below, such as intrinsic rewards and hierarchical RL, are active research areas in RL on their own. Most of the research on these topics does not consider the bi-level structure present in meta-RL. We provide a concise description of each topic in general terms but do not to provide details on the bodies of research outside of their intersection with meta-RL. For learning more about these topics, we provide references to foundational papers and surveys.

In the following, we discuss the different meta-parameterizations considered both in the single-task and multi-task settings. A summary of the methods discussed in

this section is presented in table 3.1, where the different methods are categorized by task distribution and meta-parameterization. The empty categories such as single-task meta-RL for learning hierarchical policies may be promising directions for future work. We also discuss the different outer-loop algorithms considered for many-shot meta-RL.

Learning Intrinsic Rewards The reward function of an MDP defines the task we want the agent to solve. However, the task-defining rewards may be challenging to learn from because maximizing them may not result in good exploratory behavior, e.g., when rewards are sparse [Singh et al., 2009]. One approach for making the RL problem easier is to introduce a new reward function that can guide the agent in learning how to explore. These additional rewards are called *intrinsic motivation* or *intrinsic rewards* [Aubret et al., 2019]. While intrinsic rewards are often designed manually, recently many-shot meta-RL methods have been developed to automate their design. The learned intrinsic reward functions can be represented as functions of the state and action $r^{in}(s_t, a_t)$ [Rajendran et al., 2020, Veeriah et al., 2021, Zheng et al., 2018], potential-based shaping functions $\gamma r^{in}(s_{t+1}) - r^{in}(s_t)$ [Zou et al., 2021], or functions of the entire episode so far $r^{in}(\tau_{:t})$ [Alet et al., 2020, Zheng et al., 2020]. Learning an intrinsic reward in the multi-task case can help the agent learn to explore the new environment more quickly [Alet et al., 2020, Zheng et al., 2020, Zou et al., 2021]. They can also help define skills as part of a hierarchical policy [Veeriah et al., 2021] or be used as general reward shaping in the single-task case [Zheng et al., 2018]. Beyond the standard settings, Rajendran et al. [2020] learn intrinsic rewards for practicing in extrinsic reward-free episodes in-between evaluation episodes. Finally, Meier and Mujika [2022] present an unsupervised reward learning approach, which learns complex skills in Atari games.

Learning Auxiliary Tasks In some RL problems, learning a good representation of the observations is a significant challenge for which the RL objective alone may provide poor supervision. One approach for better representation learning is introducing *auxiliary tasks*, defined as unsupervised or self-supervised objectives

optimized alongside the RL task [Jaderberg et al., 2016]. With auxiliary tasks the inner-loop objective then becomes

$$J_\phi(\mathcal{D}, \pi_\eta) = J^{\mathrm{RL}}(\mathcal{D}, \pi_\eta) + J^{\mathrm{aux}}_\phi(\mathcal{D}, \pi_\eta).$$

When a set of candidate auxiliary tasks is known, the best ones to use can be chosen by meta-learning the weight associated with each task, such that only the tasks that improve the outer-loop objective are assigned high weights [Lin et al., 2019]. Even when auxiliary tasks are not known in advance, meta-learning can be useful. Veeriah et al. [2019] use meta-gradients to discover a set of generalized value functions (GVFs) [Sutton et al., 2011], which define the prediction targets for the auxiliary loss of the policy network in the inner-loop. They show that the learned auxiliary tasks can improve sample efficiency over the base algorithm without auxiliary tasks and over handcrafted auxiliary tasks. The same approach for auxiliary task learning is used by Zahavy et al. [2020] and Flennerhag et al. [2021], who further improve the performance by tuning the hyperparameters of the inner-loop RL algorithm online. At the time of publication both achieved the state-of-the-art performance of model-free RL on the Atari benchmark [Bellemare et al., 2013].

Learning Functions that Output Hyperparameters Methods that learn functions that output hyperparameters of an inner-loop algorithm straddle the gap between hyperparameter optimization and meta-learning. Almeida et al. [2021], Flennerhag et al. [2021], Luketina et al. [2022] learn functions that take as inputs summary statistics of the inner-loop performance such as rewards and TD-error, and output the values of hyperparameters such as the λ-coefficient used in estimating returns. Furthermore, Lu et al. [2022] show that parameterizing the policy update size coefficient featured in algorithms such as proximal policy optimization (PPO) [Schulman et al., 2017] by a meta-learned function can be beneficial. The parameters of these functions are themselves optimized by meta-gradients.

Credit-Assignment Heuristics In chapter 4, we introduce a method for learning a pairwise credit assignment heuristic using meta-gradients [Zheng et al., 2021]. The proposed approach is comparable to Xu et al. [2018], who adapt the λ hyperparameter of a policy gradient algorithm online via meta-gradients. Another relevant approach, the Return Generating Model (RGM) [Wang et al., 2019b], generalizes the notion of return from exponentially discounted sum of rewards to a more flexibly weighted sum of rewards. Similar to what we do in chapter 4, they adapt the weights via meta-gradients during policy learning

Modifying the Rl Objective Directly Learning intrinsic rewards and auxiliary tasks shows that adding meta-learned terms to the RL objective can accelerate RL. These successes raise the question whether, instead of adding terms to the objectives, modifying the RL objectives directly via meta-RL can improve performance. To answer this question Houthooft et al. [2018], Oh et al. [2020], Xu et al. [2020] propose algorithms that replace the return or advantage estimator in a policy gradient algorithm with a learned function of the episode

$$\nabla_\eta J_\phi(\tau, \pi_\eta) \propto \sum_{a_t, s_t \in \tau} \nabla_\eta \log \pi_\eta(a_t|s_t) f_\phi(\tau),$$

where $f_\phi(\tau)$ is some meta-learned function of the trajectory. An alternative to replacing the advantage estimator is proposed by Kirsch et al. [2019] and Bechtle et al. [2021], who consider a deep deterministic policy gradient (DDPG)-style [Lillicrap et al., 2016] objective function, where the critic is learned via meta-RL instead of temporal difference (TD) learning.

$$\nabla_\eta J_\phi(\tau, \pi_\eta) = \sum_{a_t, s_t \in \tau} \nabla_\eta Q_\phi(s_t, \pi_\eta(a_t|s_t)),$$

where Q_ϕ is the meta-learned critic. Similar inner-loop is proposed for meta-IL by Yu et al. [2018a]. These learned RL objectives produce promising results in both multi-task and single-task meta-RL. In the multi-task setting, Oh et al. [2020] demonstrate that an objective function learned on simple tasks such as gridworld can generalize to much more complicated tasks such as Atari [Bellemare et al.,

2013]. Jackson et al. [2023] improve the generalization of the approach further by replacing the hand-designed training environments with an automated environment design component. Whereas in the single-task setting, Xu et al. [2020] show that the learned objective function can eventually outperform the standard RL algorithm (IMPALA [Espeholt et al., 2018]) it builds upon.

Learning Optimizers In most learning systems, the optimizer producing the parameter updates is manually designed. However, it is also possible to meta-learn an optimizer. Typically, the inner-loop of meta-learned optimizers conditions on losses and gradients, and outputs parameter updates. There has been some success in both many-shot and few-shot supervised learning to meta-learn the optimizer [Andrychowicz et al., 2016, Li and Malik, 2016, Ravi and Larochelle, 2017]. Some of these meta-learned optimizers use RL for meta-training [Li and Malik, 2016], and some deploy on MDPs [Chen et al., 2017]. Recently, a many-shot method has been proposed to meta-train and meta-test on MDPs, making it the first proper meta-RL method to learn an optimizer [Lan et al., 2023].

Learning Hierarchies A central problem in RL is making decisions when the consequences only become apparent after a long delay. Temporal abstraction is a potential solution to this problem [Dayan and Hinton, 1992, Nachum et al., 2018, Sutton et al., 1999]. These abstractions are commonly represented as a hierarchy consisting of a set of *options* or *skills* and a manager policy that chooses what skills to use [Bacon et al., 2017, Pateria et al., 2021, Sutton et al., 1999]. When options are available for a given environment, the manager policy can be learned with algorithms closely related to standard RL algorithms targeting the hierarchical setting. However, manually designing good options is challenging. Instead, meta-RL can be used in the discovery of options [Frans et al., 2018, Veeriah et al., 2021] by parameterizing the functions defining the objectives for learning the option with meta-learned functions. While meta-RL can help hierarchical RL the opposite is also true as a hierarchical structure can help with the central problem of meta-RL, i.e., learning policies that generalize across task distributions. Fu et al. [2020], Nam

et al. [2022] propose methods that make use of the temporal abstraction provided by hierarchical policies to solve task distributions with longer horizons.

Black-Box Meta-Learning In few-shot meta-RL, black-box methods that use RNNs or other neural networks instead of stochastic gradient descent (SGD) tend to learn faster than the SGD-based alternatives. Kirsch et al. [2021] argue that many black-box meta-RL approaches, e.g., those by Duan et al. [2016], Wang et al. [2016] cannot generalize well to unseen environments because they can easily overfit to the training environments. To combat overfitting, they introduce a specialized RNN architecture, which reuses the same RNN cell multiple times, making the RNN weights agnostic to the input and output dimensions and permutations. The proposed method requires longer trials to learn a policy for a new environment, making it a many-shot meta-RL method, but in return it can generalize to completely unseen environments.

Outer-Loop Algorithms Regardless of the inner-loop parameterization chosen, by definition, algorithms for many-shot meta-RL have to meta-learn over long task-horizons. Directly optimizing over these long task horizons is challenging because it can result in vanishing or exploding gradients and has infeasible memory requirements [Metz et al., 2021, Sutskever, 2013]. Instead, as described above, most many-shot meta-RL algorithms adopt a surrogate objective, which considers only one or a few update steps in the inner-loop [Bechtle et al., 2021, Kirsch et al., 2019, Oh et al., 2020, Rajendran et al., 2020, Veeriah et al., 2019, 2021, Zahavy et al., 2020, Zheng et al., 2018, 2020]. These algorithms use either A2C [Mnih et al., 2016]-style [Bechtle et al., 2021, Oh et al., 2020, Veeriah et al., 2021, Zheng et al., 2020] or DDPG [Lillicrap et al., 2016]-style [Kirsch et al., 2019] actor-critic objectives in the outer-loop. Flennerhag et al. [2021] present a different kind of surrogate objective, which bootstraps target parameters for the inner-loop by computing several updates ahead and then optimizing its earlier parameters to minimize distance to that later target using a chosen metric. This allows optimizing over more inner-loop updates, and with the right choice of metric, it can be used for optimizing the behavior policy

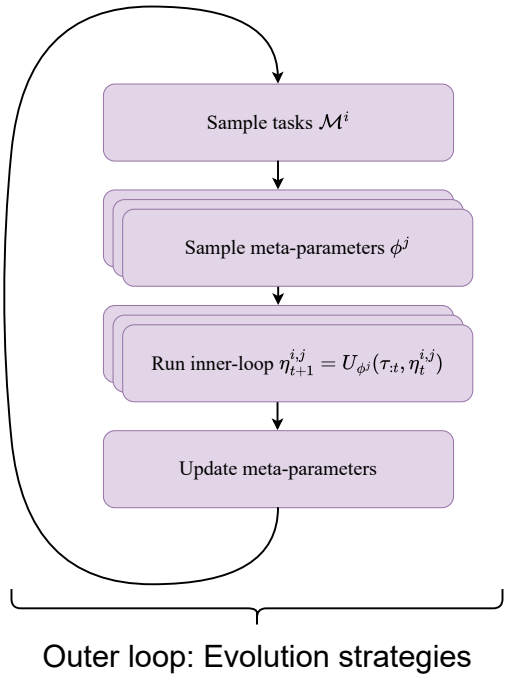


Figure 3.2: To estimate an update to the meta-parameters, ES samples a set of them and computes the inner-loop for each of them independently. This requires more evaluations of the inner-loop but can work better when the task-horizon is long.

in the inner-loop, which is difficult using a standard actor-critic objective. This surrogate objective has also been used to meta-learning how to prioritize the task distribution for policy improvement in model-based methods [Burega et al., 2022].

Alternatively *evolution strategies* (ES) [Rechenberg, 1971, Salimans et al., 2017, Wierstra et al., 2014], which are black-box optimization algorithms, are used by Houthooft et al. [2018], Kirsch et al. [2021], Lu et al. [2022]. ES works by sampling a set of parameters from a distribution, evaluating the set by running the inner-loop, and updating the parameters of the distribution. This can be seen as applying REINFORCE [Williams, 1992] on the parameter distribution. The general approach of using ES in the outer-loop is illustrated in figure 3.2. ES suffers less from the vanishing and exploding gradients problem and has more favorable memory requirements at the cost of high variance and sample complexity compared to SGD-based methods [Metz et al., 2021]. Finally, genetic algorithms [Schmidhuber, 1987] and random search are used by Alet et al. [2020], Co-Reyes et al. [2021], Garau-Luis

et al. [2022], who consider discrete parameterizations of the inner-loop objective.

3.4 Conclusion

In this chapter, we have provided an overview of meta-RL approaches, with a particular emphasis on meta-gradient methods. Reflecting the problem settings defined in chapter 2, we categorized these approaches into two settings: few-shot meta-gradient methods, which adapt policy parameters over short task horizons, and many-shot meta-gradient methods, which operate over extended horizons and aim to improve the learning efficiency of general RL algorithms. We explored how different choices in meta-parameterization and outer-loop optimization can significantly influence the performance and generalization of meta-RL agents. From adapting specific components like initializations, hyperparameters, and auxiliary tasks to designing new learning objectives and credit assignment strategies, meta-RL offers a broad and promising avenue for more efficient RL. In the next chapter, we propose a novel approach for learning credit assignment heuristics using meta-gradients, specifically focusing on pairwise credit assignment to improve policy updates. Following that, in chapter 5, we dive deep into meta-gradient estimation in outer-loop optimization. Together, these chapters build on the foundational concepts discussed here and introduce cutting-edge techniques to advance meta-RL.

3.5 Statement of Authorship

Title of Paper: A survey of meta-reinforcement learning

Publication Status

- Published
- Accepted for Publication
- Submitted for Publication ✓
- Unpublished and unsubmitted work written in a manuscript style

Publication Details: Beck, Jacob, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. "A survey of meta-reinforcement learning." *arXiv preprint* arXiv:2301.08028 (2023). Submitted for Foundations and Trends in Machine Learning.

Contribution to the paper: I was the co-lead author for the paper. I helped steer the overall research direction, surveyed the literature, and wrote significant parts of the manuscript.

4

Adaptive Pairwise Weights for Temporal Credit Assignment

Contents

4.1 Introduction

In chapter 1, we introduced meta-RL as a central topic in studying how to improve the sample efficiency of RL algorithms. More specifically, we described how meta-gradients can be used for developing adaptive RL algorithms that can use the information they gather throughout their interaction with the environment to improve the learning online. In chapter 2, we described the general structure of

meta-gradient algorithms and how to estimate the meta-gradients. In this chapter, we take a deeper look at one such adaptive RL algorithm, which uses meta-gradients to tune the credit assignment heuristic used in a policy gradient method.

We start by building intuition on why credit assignment is challenging in RL in the first place. To do so, consider the following *umbrella problem* [Osband et al., 2019], which illustrates a fundamental *temporal credit assignment* (TCA) challenge most RL algorithms face. An RL agent takes an umbrella at the start of a cloudy morning and experiences a long day at work filled with various rewards uninfluenced by the umbrella, before needing the umbrella in the rain on the way home. The agent must learn to credit the take-umbrella action in the cloudy-morning state with the very delayed reward at the end of the day, while also learning to not credit the action with the many intervening rewards, despite their occurring much closer in time. More generally, the TCA problem is how much credit or blame should an action taken in a state get for a future reward. One of the earliest and still most widely used heuristics for TCA comes from the celebrated TD(λ) [Sutton, 1988] family of algorithms, and assigns credit based on a scalar coefficient λ raised to the power of the time interval between the state-action and the reward. Note that this is a recency and frequency heuristic, in that it assigns credit based on how recently and how frequently a state-action pair has occurred prior to the reward. It is important, however, to also note that this heuristic has not in any way shown to be the "optimal" way for TCA. In particular, in the umbrella problem the action of taking the umbrella on a cloudy morning will be assigned credit for the rewards achieved during the workday early on in learning and it is only after a lot of learning that this effect will diminish. Nevertheless, the recency and frequency heuristic has been adopted in most modern RL algorithms because it is so simple to implement, with just one hyperparameter, and because it has been shown to allow for asymptotic convergence to the true value function under certain circumstances.

In this chapter, we present two new families of algorithms for addressing TCA: one that generalizes TD(λ) and a second that generalizes a Monte-Carlo algorithm. Specifically, our generalization introduces pairwise weightings that are functions of

the state in which the action was taken, *the state at the time of the reward*, and *the time interval between the two*. Of course, it isn't clear what this pairwise weight function should be, and it is too complex to be treated as a hyperparameter (in contrast to the scalar λ in TD(λ)). We develop a meta-gradient approach to learning the pairwise weight function at the same time as learning the policy of the agent. Like other meta-gradient algorithms, our algorithm has two loops: an outer-loop that periodically updates the pairwise weight function in order to optimize the usual RL loss (policy gradient loss in our case) and an inner-loop where the policy parameters are updated using the pairwise weight function set by the outer-loop.

Our contributions in this chapter can be summarized as follows

- We propose a family of algorithms that contains within it the theoretically well understood TD(λ) and Monte-Carlo algorithms.
- We show that the additional flexibility of our algorithms can yield benefit analytically in a simple illustrative example intended to build intuition and then empirically in more challenging TCA problems.
- We propose a meta-gradient algorithm to learn the pairwise-weighting function that parameterizes our family of algorithms.
- We conduct an empirical evaluation of the proposed methods aiming to answer the two questions: (1) Are the proposed pairwise weight functions able to outperform the best choice of λ and other baselines? (2) Is our meta-gradient algorithm able to learn the pairwise weight functions fast enough to be worth the extra complexity they introduce?

4.2 Related work

In chapter 3, we presented a literature survey on meta-gradient RL, covering central algorithms and parameterizations. In this section, we discuss related work on temporal credit assignment, that is specific to this chapter.

Several heuristic methods have been proposed to address the long-term credit assignment problem in RL. RUDDER [Arjona-Medina et al., 2019] trains a long short-term memory [Hochreiter and Schmidhuber, 1997, LSTM] network to predict the return of an episode given the entire state and action sequence and then conducts contribution analysis with the LSTM to redistribute rewards to state-action pairs. Synthetic Returns (SR) [Raposo et al., 2021] directly learns the association between past events and future rewards and use it as a proxy for credit assignment. Different from the predictive approach of RUDDER and SR, Temporal Value Transport (TVT) [Hung et al., 2019] augments the agent with an external memory module and utilizes the memory retrieval as a proxy for transporting future value back to related state-action pairs. We compare against TVT by using their published code, and we take inspiration from the core reward-redistribution idea from RUDDER and implement it within our policy gradient agent as a comparison baseline (because the available RUDDER code is not directly applicable). We do not compare to SR because their source code is not available.

Some recent works address counterfactual credit assignment where classic RL algorithms struggle [Harutyunyan et al., 2019, Mesnard et al., 2020, van Hasselt et al., 2020]. Although they are related to our work in that they also address the TCA problem, we do no compare to them because our work does not focus on the counterfactual aspect.

4.3 Pairwise Weights for Advantages

At the core of our contribution are new parameterizations of functions for computing advantages used in policy gradient methods. See chapter 2 for a review of advantages in policy gradient methods and TD(λ). Below we present two new estimators that are analogous in this regard to $\hat{A}^{(\lambda)}$ and $\hat{A}^{\mathrm{MC}}$.

Heuristic 1: Advantages via Pairwise Weighted Sum of TD-Errors Our first new estimator, denoted PWTD for **P**airwise **W**eighted **TD**-error, is a strict

generalization of the λ-estimator and is defined as follows:

$$\hat{A}_\phi^{\text{PWTD}}(s_t, a_t) = \sum_{k=t+1}^{T} f_\phi(s_t, s_k, k-t)\delta(s_k, a_k), \tag{4.1}$$

where $f_\phi(s_t, s_k, k-t) \in [0, 1]$, parameterized by ϕ, is the scalar weight given to the TD-error $\delta(s_k, a_k)$ as a function of the state to which credit is being assigned, the state at which the TD-error is obtained, and the time interval between the two. Note that if we choose $f(s_t, s_k, k-t) = (\gamma\lambda)^{k-t-1}$, it recovers the usual λ-estimator $\hat{A}^{(\lambda)}$.

Heuristic 2: Advantages via Pairwise Weighted Sum of Rewards Instead of generalizing from the λ-estimator, we can also generalize from the MC estimator via pairwise weighting. Specifically, the new pairwise-weighted return is defined as

$$R_\phi^{\text{PWR}}(s_t) = \sum_{k=t+1}^{T} f_\phi(s_t, s_k, k-t) r(s_k, a_k), \tag{4.2}$$

where $f_\phi(s_t, s_k, k-t) \in [0, 1]$ is the scalar weight given to the reward $r(s_k, a_k)$. The corresponding advantage estimator, denoted PWR for **P**airwise **W**eighted **R**eward, is defined as:

$$\hat{A}_\phi^{\text{PWR}}(s_t, a_t) = R_\phi^{\text{PWR}}(s_t, a_t) - v^{\text{PWR}}(s_t), \tag{4.3}$$

where $V^{\text{PWR}}(s) = \mathbb{E}_\eta[R_\phi^{\text{PWR}}(s)]$ and v^{PWR} is an approximation of V^{PWR}. Note that if we choose $f(s_t, s_k, k-t) = \gamma^{k-t-1}$, we can recover the MC estimator $\hat{A}^{\text{MC}}$.

The benefit of the additional flexibility provided by these new estimators highly depends on the choice of the pairwise weight function f. As we will demonstrate in the simple example below, the new estimators can yield lower variance and benefit policy learning if the function f captures the underlying credit assignment structure of the problem. On the other hand, the new estimators may not even be well-defined in the infinite-horizon setting if the pairwise weight function is chosen wrongly because the weighted sum of TD-errors/rewards could be unbounded. Designing a good pairwise weight function by hand is challenging because it requires both domain knowledge to capture the credit assignment structure and careful tuning to avoid harmful consequences. Thus we propose a meta-gradient algorithm to *learn* the pairwise weight function such that it benefits policy learning, as detailed in section 4.4.

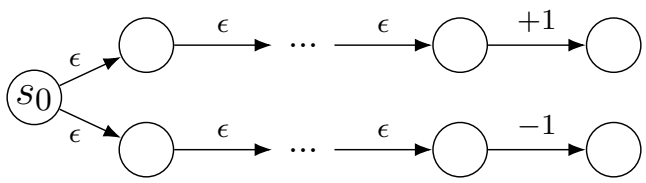


Figure 4.1: A simple illustrative MDP. The initial action determines the final reward but does not impact the intermediate rewards. The consequence of the initial action is delayed.

An Illustrative Analysis of the Benefit of the PWR Estimator Consider the simple-MDP version of the umbrella problem in figure 4.1. Each episode starts at the leftmost state, s_0, and consists of T transitions. The only choice of action is at s_0 and it determines the reward on the last transition. A noisy reward ϵ is sampled for each intermediate transition independently from a distribution with mean μ and variance $\sigma^2 > 0$. These intermediate rewards are independent of the initial action. We consider the undiscounted setting in this example. The expected return for state s_0 under policy π is

$$V(s_0) = \mathbb{E}_\pi[R(s_0)] = (T-1)\mu + \mathbb{E}_\pi[r(s_T)].$$

For any initial action a_0, the advantage is

$$A(s_0, a_0) = \mathbb{E}_\pi[R(s_0)|a_0] - V(s_0) = \mathbb{E}_\pi[r(s_T)|a_0] - \mathbb{E}_\pi[r(s_T, a_T)].$$

Consider pairwise weights for computing $\hat{A}^{\text{PWR}}(s_0, a_0)$ that place weight only on the final transition, and zero weight on the noisy intermediate rewards, capturing the notion that the intermediate rewards are not influenced by the initial action choice. More specifically, we choose f such that for any episode, $w_{0T} = 1$ and $w_{ij} = 0$ for other i and j. The shorthand w_{ij} denotes $f(s_i, s_j, j-i)$ for brevity. The expected parameterized reward sum for the initial state s_0 is

$$V^{\text{PWR}}(s_0) = \mathbb{E}_\pi[R_\phi(s_0)] = \mathbb{E}_\pi[\sum_{i=t}^{T} w_{0t} r(s_t)] = \mathbb{E}_\pi[r(s_T)].$$

If v^{PWR} is correct, for any initial action a_0, the pairwise-weighted advantage is the same as the regular advantage:

$$\begin{aligned}\mathbb{E}_\pi[\hat{A}_\phi^{\text{PWR}}(s_0, a_0)] &= \mathbb{E}_\pi[R_\phi(s_0) - v^{\text{PWR}}(s_0)|a_0] \\ &= \mathbb{E}_\pi[\sum_{t=1}^{T} w_{0t} r(s_t)] - V^{\text{PWR}}(s_0) \\ &= \mathbb{E}[r(s_T)|a_0] - \mathbb{E}_\pi[r(s_T)] = A(s_0, a_0).\end{aligned}$$

As for variance, for any initial action a_0, $[R_\phi(s_0)|a_0]$ is deterministic because of the zero weight on all the intermediate rewards and thus $\hat{A}_\phi^{\text{PWR}}(s_0, a_0)$ has zero variance. The variance of $\hat{A}^{\text{MC}}(s_0, a_0)$ on the other hand is $(T-1)\sigma^2 > 0$. Thus, in this illustrative example $\hat{A}^{\text{PWR}}$ yields an unbiased advantage estimator with far lower variance than $\hat{A}^{\text{MC}}$.

Our example exploited knowledge of the domain to set weights that would yield an unbiased advantage estimator with reduced variance, thereby providing some intuition on how a more flexible return might in principle yield benefits for learning. Of course, in general RL problems will have the umbrella problem in them to varying degrees. But how can these weights be set by the agent itself, without prior knowledge of the domain? We turn to this question next.

4.4 A Meta-Gradient Algorithm for Adapting Pairwise Weights

Recently, as we saw in chapter 3, meta-gradient methods have been developed to learn various kinds of parameters that would otherwise be set by hand or by manual hyperparameter search; We use the meta-gradient algorithm proposed by Xu et al. [2018] for training the pairwise weights. The algorithm consists of an outer-loop learner for the pairwise weight function, which is driven by a conventional policy gradient loss and an inner-loop learner driven by a policy gradient loss based on the new pairwise-weighted advantages. An overview of the algorithm is provided in appendix A. For brevity, we use $\hat{A}_\phi$ to denote $\hat{A}_\phi^{\text{PWTD}}$ or $\hat{A}_\phi^{\text{PWR}}$ unless it causes ambiguity.

Learning in the Inner-Loop. In the inner-loop, the pairwise-weighted advantage $\hat{A}_\phi$ is used to compute the policy gradient. We rewrite the gradient update from equation (2.9) with the new advantage as

$$\nabla_\eta J_\phi(\eta) = \mathbb{E}_{\tau\sim\pi_\eta}[\sum_{t=0}^{T-1} \hat{A}_\phi(s_t, a_t)\nabla_\eta \log \pi_\eta(a_t|s_t)], \tag{4.4}$$

where τ is a trajectory sampled by executing π_η. The overall update to η is

$$\nabla_\eta J^{\text{inner}}(\eta) = \nabla_\eta J_\phi(\eta) + \beta^{\mathcal{H}}\nabla_\eta \mathcal{H}(\pi_\eta), \tag{4.5}$$

where $\mathcal{H}(\eta)$ is the usual entropy regularization term [Mnih et al., 2016] and $\beta^{\mathcal{H}}$ is a mixing coefficient. We apply gradient ascent to update the policy parameters and the updated parameters are denoted by η'.

Computing $\hat{A}_\phi^{\text{PWR}}$ with equation (4.3) requires a value function predicting the expected pairwise-weighted sums of rewards. We train the value function, v_ψ with parameters ψ, along with the policy by minimizing the mean squared error between its output $v_\psi(s_t)$ and the pairwise-weighted sum of rewards $R_\phi(s_t, a_t)$. The objective for training v_ψ is

$$J_\phi^v(\psi) = \mathbb{E}_{\tau\sim\pi_\eta}[\sum_{t=0}^{T-1}(R_\phi(s_t, a_t) - v_\psi(s_t))^2]. \tag{4.6}$$

Note that $\hat{A}_\phi^{\text{PWTD}}$ does not need this extra value function.

Updating ϕ via Meta-Gradient in the Outer-Loop. To update ϕ, the parameters of the pairwise weight functions, we need to compute the gradient of the usual policy loss w.r.t. ϕ through the effect of ϕ on the inner-loop's updates to η. This corresponds to the meta-gradient described in chapter 2. Restated here for convenience, the meta-gradient is given by

$$\nabla_\phi J^{\text{outer}}(\phi) = \nabla_{\eta'} J(\eta')\nabla_\phi \eta'. \tag{4.7}$$

where,

$$\nabla_{\eta'} J(\eta') = \mathbb{E}_{\tau'\sim\pi_{\eta'}}[\sum_{i=0}^{T-1} A(s_t, a_t)\nabla_{\eta'} \log \pi_{\eta'}(a_t|s_t)], \tag{4.8}$$

τ' is another trajectory sampled by executing the updated policy $\pi_{\eta'}$ and A is the regular advantage.

Note that we need two trajectories, τ and τ', to make one update to the meta-parameters ϕ. The policy parameters η are updated after collecting trajectory τ. The next trajectory τ' is collected using the updated parameters η'. The ϕ-parameters are updated on τ'. In order to make more efficient use of the data, we follow Xu et al. [2018] and reuse the second trajectory τ' in the next iteration as the trajectory for updating η. In practice, we use modern auto-differentiation tools to compute equation (4.7) without applying the chain rule explicitly. Computing the regular advantage requires a value function for the regular return. This value function is parameterized by ξ and updated to minimize the squared loss analogously to v_ξ.

4.5 Experiments

In this section, we present three sets of experiments to investigate the effectiveness of the proposed advantage estimators and the meta-gradient algorithm to learn them. In section 4.5.1, we show experiments in simple tabular MDPs that allow visualization of the pairwise weights learned by Meta-PWTD and -PWR. The results show that the meta-gradient adaptation both *increases* and *decreases* weights in a way that can be interpreted as reflecting explicit credit assignment and variance reduction. In the second set of experiments presented in section 4.5.2, we test Meta-PWTD and -PWR with neural networks in the benchmark credit assignment task *Key-to-Door* [Hung et al., 2019]. We show that Meta-PWTD and -PWR outperform several existing methods for directly addressing credit assignment, as well as TD(λ) methods, and show again that the learned weights reflect domain structure in a sensible way. In the third set, in section 4.5.3, we evaluate Meta-PWTD and -PWR in two benchmark RL domains, `bsuite` [Osband et al., 2019] and Atari, and show that our methods do not hinder learning when environments do not pose idealized long-term credit assignment challenges.

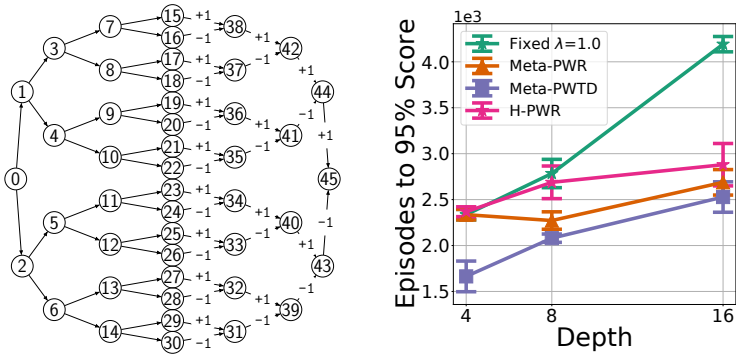


Figure 4.2: (Left) Depth 8 DAG environment with choice of two actions at each state and rewards along transitions. (Right) Learning performance of regular return, handcrafted weights, and fixed meta-learned weights. Results are averaged over 5 independent runs. Low is good.

4.5.1 Learned Pairwise Weights in a Simple MDP

Consider the environment represented as a directed acyclic graph (DAG) in figure 4.2 (left). In each state in the left part of the DAG (states 0–14, the *first phase*), the agent chooses one of two actions but receives no reward. In the remaining states (states 15–44, the *second phase*), the agent has only one action available and it receives a reward of $+1$ or -1 at each transition. Crucially, the rewards the agent obtains in the second phase are a consequence of the action choices in the first phase because they determine which states are encountered in the second phase. There is an interesting credit assignment problem with a nested structure; for example, the action chosen at state 1 determines the reward received later upon transition into state 44. We refer to this environment as the *Depth* 8 *DAG* and also report results below for depths 4 and 16.

In the DAG environments, we use tabular policy, value function, and meta-parameter representations. The parameters η, ψ, ξ, and ϕ represent the policy, baseline for the weighted return, baseline for the regular return, and meta-parameters respectively. The meta-parameters ϕ are a $|\mathcal{S}| \times |\mathcal{S}|$ matrix. The entry on the ith row and the jth column defines the pairwise weight for computing the contribution of reward at state j to the return at state i. A sigmoid function is used to bound the weights to $[0, 1]$, and the meta-parameters are initialized so that the pairwise weights are close to 0.5. Note that even in a tabular domain such as the DAG,

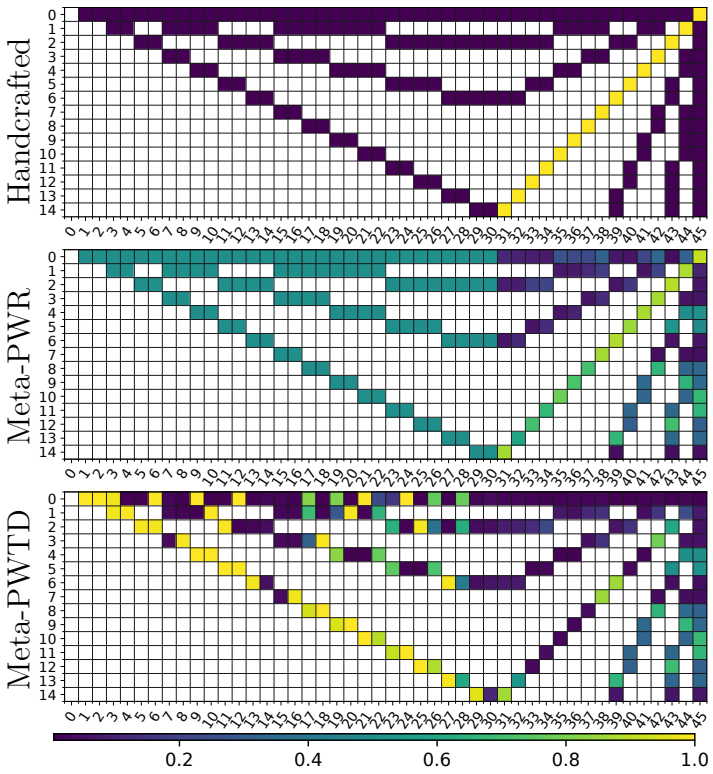


Figure 4.3: inner-loop-reset weight visualization: Top: Handcrafted pairwise weights for Depth 8 DAG; rows and columns correspond to states in Figure 4.2. Middle: Meta-learned weights for Depth 8 DAG for Meta-PWR and Bottom: Meta-PWTD.

setting the credit assignment weights by random search would be infeasible due to the number of possible weight combinations. This problem is exacerbated by larger domains discussed in the following sections. For this reason, the meta-gradient algorithm is a promising candidate for setting the weights.

Visualizing the Learned Weights via Inner-Loop Reset To clearly see the most effective weights learned with the meta-gradient for a random policy, we repeatedly reset the policy parameters to a random initialization while continuing to train the meta-parameters until convergence. More specifically: the meta-parameters ϕ are trained repeatedly by randomly initializing η, ψ, and ξ and running the inner-loop for 16 updates for each outer-loop update. Following existing work in meta-gradient [Veeriah et al., 2019, Zheng et al., 2020], the outer-loop objective is evaluated on all 16 trajectories sampled with the updated policies. The gradient of the outer-loop objective on the ith trajectory with respect to ϕ is

backpropagated through all of the preceding updates to η. The hyperparameters for this experiment are provided in appendix A.

What pairwise weights would accelerate learning in this domain? Figure 4.2 (top) visualizes a set of *handcrafted* weights for $\hat{A}^{\text{PWR}}$ in the Depth 8 DAG; each row in the grid represents the state in which an action advantage is estimated, and each column the state in which a future reward is experienced. For each state pair (s_i, s_j) the weight is 1 (yellow) only if the reward at s_j depends on the action choice at s_i, else it is zero (dark purple; the white pairs are unreachable). Figure 4.2 (middle) shows the corresponding weights learned by Meta-PWR. Importantly, the learned pairwise weights have been *increased* for those state pairs in which the handcrafted weights are 1 and have been *decreased* for those state pairs in which the handcrafted weights are 0. As in the analysis of the simple domain in section 4.3, these weights will result in lower variance advantage estimates.

The same reset-training procedure was applied to A^{PWTD}. Figure 4.2 (bottom) visualizes the resulting weights. Since the TD-errors depend on the value function which are nonstationary during agent learning, we expect different weights to emerge at different points in training; the presented weights are but one snapshot. Regardless of the nonstationarity, a clear contrast to reward weighting can be seen: high weights are placed on transitions in the first phase of the DAG, which yield no rewards—because the TD-errors at these transitions do provide signal once the value function begins to be learned. In appendix A, we explicitly probe the adaptation of A^{PWTD} to different points in learning by modifying the value function in reset experiments, and show that the weights indeed adapt sensibly to differences in the accuracy of the value function.

Evaluation of the Learned Pairwise Weights After the η-reset training of the pairwise-weights completed, we used them to train a new set of η parameters, fixing the pairwise weights during learning. Figure 4.2 (right) shows the number of episodes to reach 95% of the maximum score in each DAG, for policies trained with regular returns, handcrafted weights (H-PWR), and meta-learned weights.

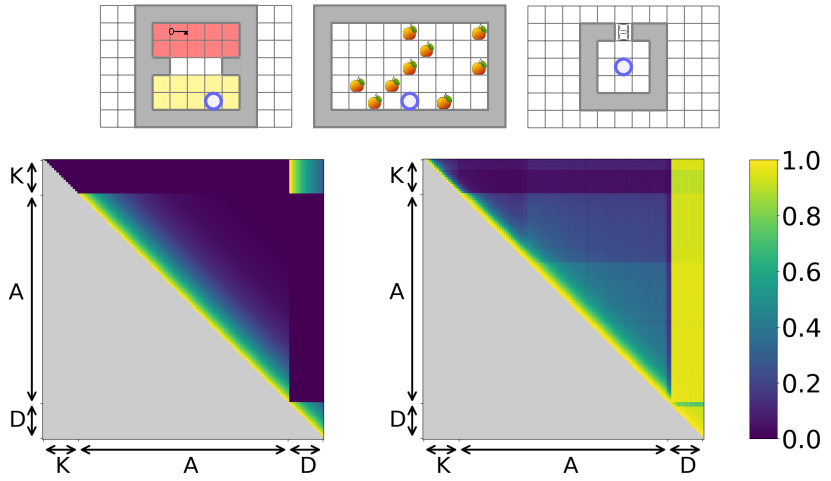


Figure 4.4: (Top) The three phases in KtD. The blue circle denotes the agent. (Bottom left) Visualization of Handcrafted weights in the KtD experiment. (Bottom right) Weights learned by Meta-PWR in the $\mu = 5, \sigma = 5$ configuration.

Using the meta-learned weights learned as fast as (indeed faster than) using the handcrafted weights, and both were faster than the regular returns, with the gap increasing for larger DAG-depth. We conjecture that the learned weights performed even better than the handcrafted weights because the learned weights adapted to the dynamics of the inner-loop policy learning procedure whereas the handcrafted weights did not. Learning curves presented in appendix A show that all method achieved the optimal performance in the end.

4.5.2 Key-to-Door Experiments

We evaluated Meta-PWTD and -PWR the Key-to-Door (KtD) environment [Hung et al., 2019] that is an elaborate umbrella problem that was designed to show-off the TVT algorithm's ability to solve TCA. We varied properties of the domain to vary the credit assignment challenge. We compared the learning performance of our algorithms to a version of $\hat{A}^{\text{PWR}}$ that uses fixed handcrafted pairwise weights and no meta-gradient adaptation, as well as to the following **five** baselines (see related work in section 4.2): **(a)** best fixed-λ: Actor-Critic (A2C) [Mnih et al., 2016] with a best fixed λ found via hyperparameter search; **(b)** TVT [Hung et al., 2019] (using the code accompanying the paper); **(c)** A2C-RR: a reward redistribution method *inspired* by RUDDER [Arjona-Medina et al., 2019]; **(d)** Meta-$\lambda(s)$ [Xu

et al., 2018]: meta-learning a state-dependent function $\lambda(s)$ for λ-returns; and **(e)** RGM [Wang et al., 2019b]: meta-learning a *single* set of weights for generating returns as a linear combination of rewards.

Environment and Parametric Variation KtD is a fixed-horizon episodic task where each episode consists of three phases (figure 4.4 top). In the *Key phase* (15 steps in duration), there is no reward and the agent must navigate to the key to collect it. In the *Apple phase* (90 steps in duration), the agent collects apples; apples disappear once collected. Each apple yields a noisy reward with mean μ and variance σ^2. In the *Door phase* (15 steps in duration), the agent starts at the center of a room with a door but can open the door only if it has collected the key earlier. Opening the door yields a reward of 10. Crucially, picking up the key or not has no bearing on the ability to collect apple rewards. The apples are the noisy rewards that *distract the agent from learning that picking up the key early on leads to a door-reward later.* In our experiments, we evaluate methods on 9 different environments representing combinations of 3 levels of apple reward mean and 3 levels of apple reward variance.

Implementation The agent observes the top-down view of the current phase rendered in RGB and a binary variable indicating if the agent collected the key or not. The policy (η) and the value functions (ψ and ξ) are implemented by separate convolutional neural networks. The *meta-network* (ϕ) computes the pairwise weight w_{ij} as follows: First, it embeds the observations s_i and s_j and the time difference $(j - i)$ into separate latent vectors. Then it takes the element-wise product of these three vectors to fuse them into a vector h_{ij}. Finally it maps h_{ij} to a scalar output that is bounded to $[0, 1]$ by sigmoid. We tuned the hyperparameters for each method on the mid-level configuration $\langle \mu = 5, \sigma = 25 \rangle$ and kept them fixed for the other 8 configurations. Each method has a distinct set of parameters (e.g. outer-loop learning rates, λ). We referred to the original papers for the parameter ranges searched over. More details are in appendix A.

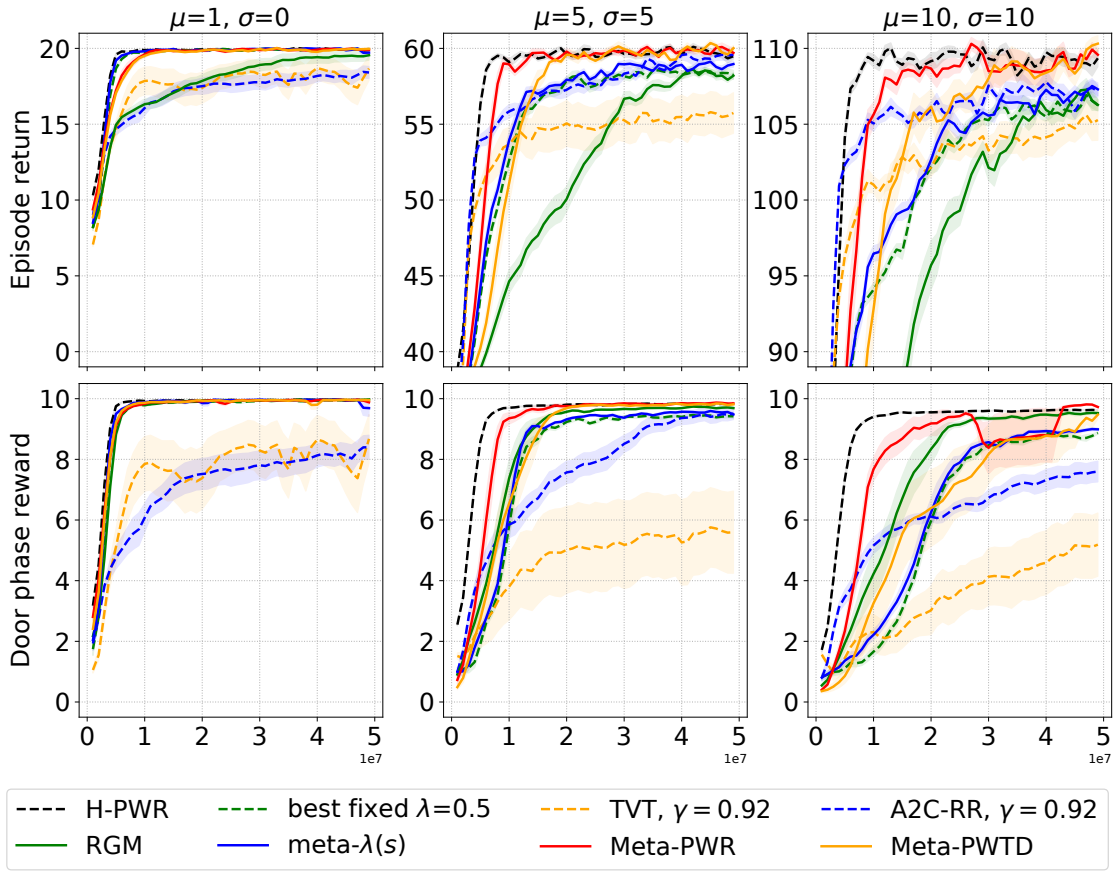


Figure 4.5: Learning curves for the KtD domain. Each column corresponds to a different configuration. The x-axis denotes the number of frames. The y-axis denotes the episode return in top row and the door phase reward in bottom row. The solid curves show the average over 10 independent runs and the shaded area shows the standard errors.

Empirical Results Figure 4.5 presents learning curves for Meta-PWTD, Meta-PWR, and baselines in three KtD configurations (the remaining configurations are in the appendix). Learning curves are shown separately for the *total episode return* and the *door phase reward*, the latter a measure of success at the long-term credit assignment. Not unexpectedly, H-PWR which uses handcrafted pairwise weights performs the best. The gap in performance between H-PWR and the best fixed-λ shows that this domain provides a credit assignment challenge that the pairwise-weighted advantage estimate can help with. The TVT and A2C-RR methods used a low discount factor and so relied solely on their heuristics for learning to pick up the key, but neither appears to enable fast learning in this domain. In the door phase, Meta-PWR is generally the fastest learner after H-PWR. Meta-PWTD, though

	Catch	Catch Noise	Catch Scale	Umbr. Length	Umbr. Distract	Cartpole	Discount Chain
A2C	5975	42221	56800	38050	37524	76874	3554
A2C-RR	**5950**	42295	57033	38083	37433	71506	3548
RGM	7849	48268	54421	40397	40159	119102	2444
Meta-PWTD	6096	**41106**	**48199**	**37973**	37226	65945	1040
Meta-PWR	5967	43076	49361	38168	**36554**	**61752**	**161**

Table 4.1: Total regret on selected `bsuite` domains (low is good).

slower, achieves optimal performance. Although RGM performs third best in the door phase, it does not perform well overall, suggesting that the inflexibility of its single set of reward weights (vs. pairwise of Meta-PWR) forces a trade off between short and long-term credit assignment. In summary, Meta-PWR outperforms all the other methods and Meta-PWTD is comparable to the baselines.

Figure 4.4 presents a visualization of the handcrafted weights for H-PWR (bottom left) and weights learned by Meta-PWR (bottom right). In each heatmap, the element on the i-th row and the j-th column denotes w_{ij}, the pairwise weight for computing the contribution of the reward upon transition to the j-th state to the return at the i-th state in the episode. In the heatmap of the handcrafted weights, the top-right area has non-zero weights because the rewards in the door phase depend on the actions selected in the key phase. The weights in the remaining part of the top rows are zero because those rewards do not depend on the the actions in the key phase. For the same reason, the weights in the middle-right area are zero as well. The weights in the rest of the area resemble the exponentially discounted weights with a discount factor of 0.92. This steep discounting helps fast learning of collecting apples. The learned weights largely resemble the handcrafted weights, which indicate that the meta-gradient procedure was able to simultaneously learn (1) the important rewards for the key phase are in the door phase, and (2) a quick-discounting set of weights within the apple phase that allows faster learning of collecting apples.

4.5.3 Experiments on Standard RL Benchmarks

Both the DAG and KtD domains are idealized credit assignment problems. However, in domains outside this idealized class, Meta-PWTD and -PWR may learn slower than baseline methods due to the additional complexity they introduce. To

evaluate this possibility we compared them to baseline methods on `bsuite` [Osband et al., 2019] and Atari [Bellemare et al., 2013], both standard RL benchmarks. For these experiments, we did not compare to Meta-$\lambda(s)$ because it performed similarly to the fixed-λ baseline in previous experiments as noted in the original paper [Xu et al., 2018].

`bsuite` is a set of unit-tests for RL agents: each domain tests one or more specific RL-challenges, such as exploration, memory, and credit assignment, and each contains several versions varying in difficulty. We selected all domains that are tagged by "credit assignment" and at least one other challenge. These domains are not designed solely as idealized credit assignment problems. We ran all methods for $100K$ episodes in each domain except *Cartpole*, which we ran for $50K$ episodes. Each run was repeated 3 times with different random seeds. Table 4.1 shows the total regret. Overall Meta-PWTD or -PWR achieved the lowest total regret in all domains except for *Catch*. It shows that Meta-PWTD and -PWR perform better than or comparably to the baseline methods even in domains without the idealized umbrella-like TCA structure.

To test scalability to high-dimensional environments, we conducted experiments on Atari. Atari games often have long episodes of more than 1000 steps thus episode truncation is required. However, the returns in RGM and Meta-PWR are not in a recursive additive form thus the common way of correcting truncated trajectories by bootstrapping from the value function is not applicable. TVT also requires full episodes for value transportation. Therefore, we excluded RGM, TVT, and Meta-PWR and only ran Meta-PWTD, A2C-RR and A2C. For each method we conducted hyperparameter search on a subset of 6 games and ran each method on 49 games with the fixed set of hyperparameters; see appendix for details. An important hyperparameter for the A2C baseline is λ, which was set to 0.95.

Figure 4.6 (inset) shows the median human-normalized score during training. Meta-PWTD performed slightly better than A2C over the entire period, and both performed better than A2C-RR Figure 4.6 shows the relative performance of Meta-PWTD over A2C. Meta-PWTD outperforms A2C in 30 games, underperforms in 14,

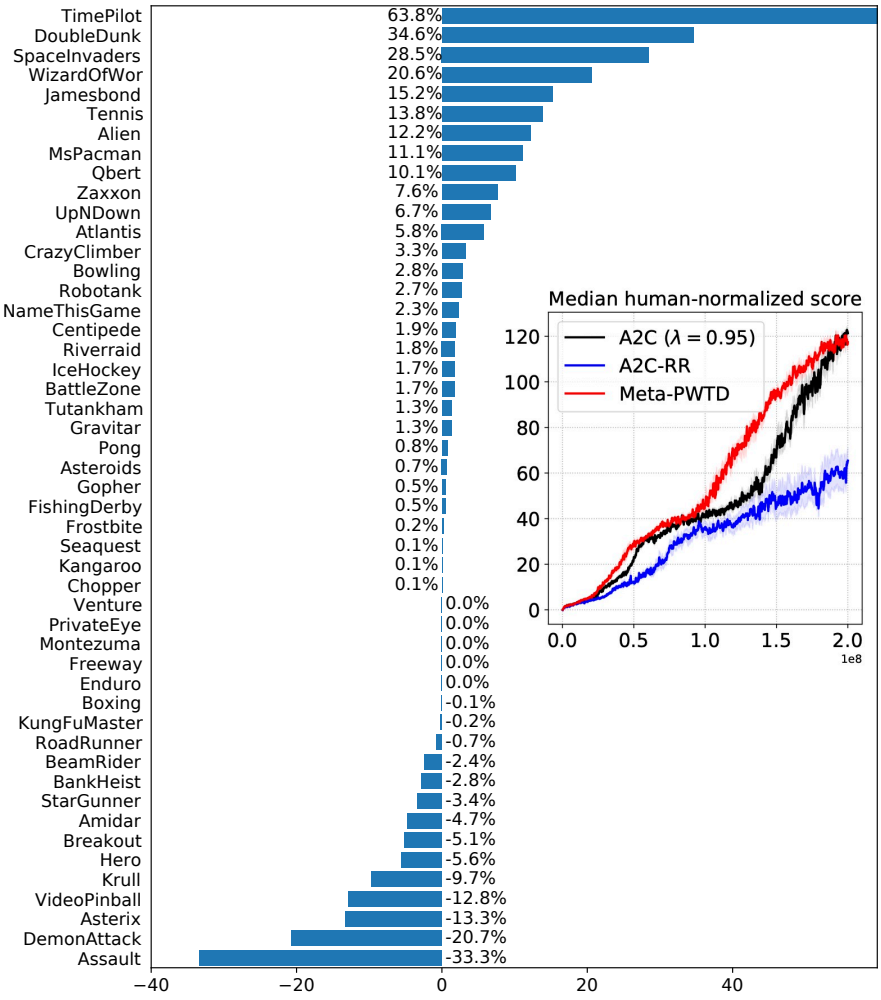


Figure 4.6: Relative performance of Meta-PWTD over A2C ($\lambda = 0.95$). All scores are averaged over 5 independent runs with different random seeds. Inset: Learning curves of median human normalized score of all 49 Atari games. Shaded area shows the standard error over 5 runs.

and ties in 5. These results show that Meta-PWTD can scale to high-dimensional environments like Atari. We conjecture that Meta-PWTD provides a benefit in games with embedded umbrella problems but this is hard to verify directly.

4.6 Conclusion

In this chapter, we introduced two new advantage estimators with pairwise weight functions as parameters, designed for use in policy gradient algorithms, along with a meta-gradient algorithm to learn these pairwise weights. Our simple analysis and empirical work confirmed that the additional flexibility provided by these advantage

estimators is beneficial, particularly in domains with delayed consequences of actions, such as umbrella-like problems. Moreover, we demonstrated that the method can be used even in large-scale environments like Atari. These findings complement the broader theme of enhancing sample efficiency in RL, as discussed in chapter 1. In the next chapter, we will return to meta-gradient algorithms but instead of proposing new applications for meta-gradients we focus on better estimation of them.

4.7 Statement of Authorship

Title of Paper: Adaptive pairwise weights for temporal credit assignment

Publication Status

- Published ✓
- Accepted for Publication
- Submitted for Publication
- Unpublished and unsubmitted work written in a manuscript style

Publication Details: Zheng, Zeyu, Risto Vuorio, Richard Lewis, and Satinder Singh. "Adaptive pairwise weights for temporal credit assignment." *In Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, pp. 9225-9232. 2022.

Concurrent submission details: The paper has been accepted as part of the thesis: Zheng, Zeyu. "Advances in Deep Reinforcement Learning: Intrinsic Rewards, Temporal Credit Assignment, State Representations, and Value-equivalent Models" PhD diss., University of Michigan, 2022.

Contribution to the paper: I was the co-lead author for the paper. I helped steer the overall research direction, collaborated on implementing the algorithms, conducted experiments, and helped write the manuscript.

5

An Investigation of the Bias-Variance Tradeoff in Meta-Gradients

Contents

5.1 Introduction

In the previous chapter, we introduced an adaptive credit assignment heuristic, which we tuned online using meta-gradients. This and the many other meta-

gradient algorithms surveyed in chapter chapter 3 are promising approaches for more sample efficient RL. However, in order for these algorithms to work well, they need high quality estimates of the meta-gradients. While meta-gradient estimation has attracted significant attention, as we saw in chapter 3, it has mostly focused on the few-shot meta-RL setting. Furthermore, the previous work on meta-gradient estimation has resulted in some amount of confusion about bias of the estimators and best practices. To help fix this, in this chapter, we present a study of the bias-variance tradeoff in meta-gradients in the many-shot setting.

We start off by tackling a confusion about the bias in the meta-gradient estimators. Computing the meta-gradient requires differentiating through the agent's parameter update, which may itself include an estimated policy gradient. Foerster et al. [2018] and others have claimed that an estimate of the *expected* policy Hessian should be included in the meta-gradient estimator, and provide methods for calculating such an estimate or reducing its variance. In this chapter, we present the surprising result that using an estimate of the expected Hessian in the meta-gradient estimator actually *adds* bias, which can degrade performance. Such a term would only make sense if the agent made an *expected* policy gradient update, rather than the sampled update that must be taken using finite data. Instead, the correct meta-gradient of a standard policy gradient update can be computed by straightforward backpropagation of a surrogate loss, as is typical in RL. This focus on estimating expected Hessians may have distracted the community from the challenges in meta-gradient estimation in the many-shot setting described in chapter 2.

Therefore, we continue with an investigation of the bias from truncating the meta-gradient estimator, which is a problem specific to the many-shot setting. In the few-shot setting, the meta-gradient can be computed via backpropagation over the whole optimization trajectory of the inner learning problem. Due to computational constraints, this is not feasible in the many-shot setting. A common approach is to compute the meta-gradient over a truncated optimization horizon. As a result of truncation, neither the direct meta-gradient nor the sampling correction may be calculated exactly. It is therefore unclear how the truncated sampling correction

affects the bias and variance of the meta-gradient estimation. In this chapter, we study how bias and variance may be traded off by varying the truncation horizon and the weight given to the sampling correction. We find that, while sampling correction does not uniformly reduce the bias of truncated estimators, it nevertheless yields convergence to better local optima. The cost of using the sampling correction is increased variance.

These considerations highlight that meta-gradient estimation, particularly in the many-shot meta-RL setting, is a complex challenge. This raises the question: are there simpler approaches for optimizing meta-parameters? One alternative is to use black-box methods like ES [Rechenberg, 1971, Schwefel, 1977]. On one hand, as a black box method, ES does not require new derivations for handling the changing data distribution and it has favorable memory requirements when used with long optimization horizons. On the other, its sample complexity grows with the number of parameters being optimized. We empirically relate the bias and variance of backpropagation based meta-gradient estimators to those of ES and find a cross-over point, where the variance of the sampling corrected meta-gradient becomes higher than that of ES. We also compare the different estimators in a many-shot setting in practice and find that none of them are good enough for training complex meta-parametrizations.

Our contributions are as follows.

- We show mathematically why using the expected policy Hessian estimator is incorrect and demonstrate that the resulting bias harms performance in practice.

- We show how the variance of the sampling correction for meta-gradients grows counter intuitively, how it works in the truncated optimization setting, and how standard advantage estimation fails for the sampling correction.

- We characterize the bias-variance tradeoff due to truncated optimization and the sampling correction in an empirical study, relating existing approaches to

each other and showing how to interpolate between them. We also compare the backpropagation-based estimators to ES.

5.2 Background

In this chapter, we assume the many-shot single-task meta-RL setting described in section 2.2.2. To briefly recap the setting, the inner-loop is a policy gradient algorithm with meta-parameters ϕ producing a sequence of task-horizon H policies parameterized by η^h. The outer-loop updates the meta-parameters to maximize the return of each of the policies produced by the inner-loop as defined by the meta-RL objective equation (2.16). All of the derivations presented in this chapter are straightforward to apply to the multi-task meta-RL setting as well.

When the task-horizon H is large, computing the exact meta-gradient with backpropagation is challenging due to limited memory and the rugged objective landscape [Metz et al., 2019]. Hence, backpropagation along the trajectory of updates is commonly truncated to a window of optimization steps much shorter than H. Computing the meta-gradient over such truncation windows results in bias. Nonetheless, this approach has been employed successfully in practical meta-RL algorithms, e.g. Oh et al. [2020]. In the multi-task meta-RL setting, the truncation is needed for computational convenience. However, in the single-task meta-RL setting, where the meta-gradients are used for tuning an RL algorithm online, the truncation is necessary because the outer-loop can't wait for the inner-loop optimization to conclude.

5.2.1 Estimating the Hessian of Policy Value

The standard approach for estimating the first derivative of $J(\eta)$ is to construct a surrogate loss of the form $\sum_{\tau \in \mathcal{D}} \sum_{s,a \in \tau} \log \pi_\eta(a|s) R(\tau)$ and differentiate it w.r.t. η. Foerster et al. [2018] show that naively differentiating this surrogate loss a second time does not compute an unbiased estimate of the Hessian $\nabla^2_\eta J(\eta)$. They propose a new surrogate objective called DiCE that may be differentiated any number of times to compute estimators of any-order derivatives of $J(\eta)$. Subsequent

work [Farquhar et al., 2019, Liu et al., 2019, Mao et al., 2019, Rothfuss et al., 2018, Tang et al., 2021] builds on this approach or proposes alternative strategies for estimating this Hessian, which is assumed to be important for meta-gradient estimation. However, contrary to the claims made by Foerster et al. [2018] and others, we show below that when estimating the Hessian for meta-gradients in practice, using an estimate of $\nabla^2_\eta J(\eta)$ always adds bias and has high variance whereas the naive approach does not add bias.

5.3 Bias and Variance of Meta-Gradient Estimators

In this section, we investigate the bias and variance of meta-gradients due to the Hessian estimation and sampling corrections in theory. We first show the surprising result that including the Hessian of the *expected* inner objective adds rather than removes bias. Then, we explore three key considerations relating to the sampling correction terms, which are missing from most current meta-gradient algorithms and have not been discussed widely in the literature. First, we show how the variance of the sampling correction terms grows counter intuitively with the batch size of the inner loop, and present a weighting scheme for mitigating variance. Second, we discuss the bias from sampling corrections in the truncated optimization setting. Third, we show that standard advantage estimation with a state-dependent baseline results in bias when used with the sampling correction terms.

5.3.1 Estimating the Hessian of the Inner-Loop Objective

If the inner-loop update function is gradient-based, then the term $\nabla_{\eta^h} U_\phi(\eta^h, \mathcal{D}^h)$ in equation 2.19 induces a second-order derivative: the Hessian of the objective that is differentiated in the inner-loop. If this inner-loop objective was the expected policy value $J(\eta)$, then this Hessian is exactly the Hessian of policy value discussed in section 5.2.1:

$$\nabla^2_\eta J(\eta) = \mathbb{E}_{\tau\sim p(\tau;\eta)}\left[\left(\nabla_\eta \log \pi_\eta(\tau) \nabla_\eta \log \pi_\eta(\tau)^\top + \nabla^2_\eta \log \pi_\eta(\tau)\right) R(\tau)\right]. \quad (5.1)$$

The works discussed in section 5.2.1 either directly estimate this quantity through sampling or propose lower-variance, but biased, estimators. Regardless, they all refer to equation 5.1 as the desired quantity.

However, in almost all practical cases, the *expected* policy gradient $\nabla_\eta J(\eta)$ is intractable and thus cannot be used in the inner loop. Instead, we must use the gradient of the *sampled* objective $\nabla_\eta J(\eta, \mathcal{D})$ given by equation 2.10. The update function is then a deterministic function of its inputs, which now include the data. The Hessian of this sampled objective is consequently easy to compute with reverse mode automatic differentiation, simply by backpropagating through the computation graph of the sequence of updates in equation (2.17). This is indeed what many meta-gradient implementations do in practice [Zahavy et al., 2020, Zheng et al., 2020]. Computing the Hessian of the sampled objective gives:

$$\nabla_\eta^2 J(\eta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \nabla_\eta^2 \log \pi_\eta(\tau) R(\tau). \tag{5.2}$$

The term $\nabla_\eta \log \pi_\eta(\tau) \nabla_\eta \log \pi_\eta(\tau)^\top R(\tau) \in \mathbb{R}^{d_\eta \times d_\eta}$ does not appear in this stochastic Hessian. Estimates of the Hessian of the expected objective thus include this spurious term which introduces bias, except in edge cases for which it vanishes, such as MDPs with no rewards. Therefore, the meta-gradient estimators discussed in section 5.2.1, which estimate the Hessian by equation (5.1), are biased in practice. In section 5.4, we illustrate the bias and variance resulting from using these estimators in practice.

5.3.2 Variance due to the Sampling Correction

In section 2.2.1, we introduced the unbiased meta-gradient estimator, which includes the sampling correction terms of the form $\nabla_\phi \eta^j \nabla_{\eta^j} \log p_\eta^j(\mathcal{D}^j) R(\tau)$ where $\log p_\eta^j(\mathcal{D}^j) = \sum_{\tau \in \mathcal{D}^j} \log p_\eta^j(\tau)$. These terms include the gradient of the log probability of the entire batch of samples $\mathcal{D}^j$, and so involve a sum over the batch elements.

Intuitively, in normal policy gradient RL the probability of sampling a particular reward can only depend on the trajectory leading to that reward as the policy parameters η are fixed. In meta-gradient meta-RL, by contrast, the parameters η^j vary, and depend on every trajectory used by the update function up to iteration j.

The sampling correction accounts for these dependencies. Unfortunately, this means the variance of the sampling correction counter intuitively grows with the batch size of the inner loop. The variance is of course still reduced by increasing the meta-batch size in the outer-loop.

In normal RL, although the dependencies are restricted to a single trajectory, a discount factor is still typically used to downweight the credit assigned to temporally distant actions. This reduces variance, but introduces bias with respect to the undiscounted objective. Similarly, we may downweight the sampling correction terms to trade off bias and variance in meta-gradient meta-RL. One strategy for enabling this trade-off is to uniformly weight the sampling correction terms by a coefficient ξ:

$$\sum_{h=0}^{H} \frac{1}{|\mathcal{D}|}\left(\xi \sum_{j=0}^{h-1} \nabla_{\phi}\eta^{j}\nabla_{\eta^{j}} \log p_{\eta^{j}}(\mathcal{D}^{j}) \sum_{\tau\in\mathcal{D}^{h}} R(\tau) + \sum_{\tau\in\mathcal{D}^{h}} \nabla_{\phi}\eta^{h}\nabla_{\eta^{h}} \log p_{\eta^{h}}(\tau)R(\tau)\right). \tag{5.3}$$

This sampling correction coefficient is related to the ξ hyperparameter of E-MAML [Stadie et al., 2018], but we do not separately divide by the batch size, which is done in implementations of E-MAML though not motivated in the paper. Therefore, the meta-gradient estimator is unbiased when our $\xi = 1.0$. We investigate the bias-variance tradeoff induced by this ξ in section 5.4. The coefficient ξ is only one possible scheme to trade off bias and variance. Motivated by analogy to the usual discounting procedure in RL, we consider an exponential discounting in appendix B.2.

5.3.3 Sampling Corrections in Truncated Optimization

To the best of our knowledge none of the many-shot meta-RL algorithms using truncated backpropagation employ any form of sampling correction, and it is not known how the sampling correction works in the truncated setting. The truncation changes the terms in equation 2.19, making it a biased estimator. Since the sampling correction terms are a product of the derivative in equation (2.19) and the gradient of the log probability of a batch, they also become biased when truncation is used. Because of the complex dynamics of the optimization process, we have no reason to believe that the bias from truncating the backpropagation decreases

monotonically with the truncation length. Therefore, it is possible that using the sampling correction on a truncated meta-gradient estimator increases rather than decreases the bias. In addition to changing the derivatives, approximating the sum over H in equation 5.3 with a sum over a shorter window causes bias by modifying the surrogate loss even before backpropagation. We investigate the effect of sampling correction on the truncated estimators empirically in section 5.4.

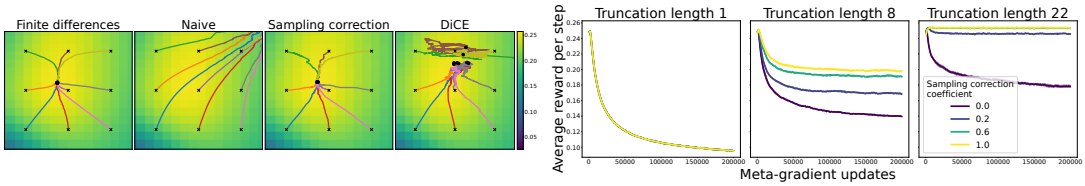
5.3.4 Advantage Estimation for Sampling Corrections

Intuitively, we would like to use a standard advantage estimator, such as GAE proposed by Schulman et al. [2015], for the sampling correction terms in equation 2.18. However, we now show that using standard advantage estimation with the sampling correction results in bias, due to dependencies between the previous policies on the update trajectory and the value estimates of the updated policies.

As discussed in chapter 2, the policy gradient via the surrogate loss with a simple advantage estimator is given by $\nabla_\eta J(\eta) \propto \mathbb{E}_{s,a\sim p_\eta(s,a)}[\nabla_\eta \log \pi_\eta(a|s)(Q(s,a) - b(s))]$. The simple advantage estimator $Q(s,a) - b(s)$ is unbiased because $b(s)$ is independent of a given s from which follows $\mathbb{E}_a[\nabla_\eta \log \pi_\eta(a|s) b(s)] = 0$. In equation 2.18, it is tempting to replace $R(\tau)$ with an advantage estimator given by $R(\tau) - \hat{V}(\tau)$, where $\hat{V}(\tau)$ is an estimate of the value of the policy in the first state of the trajectory τ and $\hat{V}$ is trained on the data $\mathcal{D}^i$ for $0 \leq i < h$. This would allow sharing the value estimator between the inner-loop and outer-loop. To get an unbiased gradient estimate, we need $\mathbb{E}_{\mathcal{D}^i, \tau\sim p_{\eta^h}(\tau)}[\nabla_\phi \log p_{\eta^i}(\mathcal{D}^i)\hat{V}(\tau)]$ for $0 \leq i < h$ to equal 0. However, this can be nonzero and thus lead to bias because the value function estimator $\hat{V}$ itself depends on the data $\mathcal{D}^i$. More generally, any $b(s)$, which depends on $\mathcal{D}^i$ leads to bias. In appendix B.3, we demonstrate empirically the bias due to the dependence of the value estimator on $\mathcal{D}^i$.

5.4 Experiments

In this section, we conduct experiments to test the bias-variance tradeoff in practice. We first consider a bandit setting to illustrate the sources of bias and the bias-



(a) Meta-parameter trajectories (b) Learning curves

Figure 5.1: (a) Comparing meta-gradient estimators in the bandit setting. In all panels, the x and y axes correspond to the two meta-parameters defining the learning rate schedule, the background shows the average return with the meta-parameter value sampled at the center of the cell, the crosses show the nine initial meta-parameter values, and the lines terminating in circles show the trajectories the parameters take during meta-training. **(b)** Meta-learning curves for the meta-gradient estimators in the bandit setting with truncated meta-optimization horizons. The shading of the curves shows the standard error across seeds.

variance tradeoff in a simple case. Then, to verify that the findings from the theory and the bandit experiments apply to the full RL problem, we consider meta-learning in a gridworld.

In the bandit setting, we train agents with REINFORCE [Williams, 1992] for multiple update steps on bandits with randomly sampled arm rewards. The meta-learning problem is to learn separate learning rates for the early and late parts of the inner learning. Hence, these two meta-parameters define a learning rate schedule; intuitively, the agent must meta-learn an optimal trade-off of exploration and exploitation for a given distribution of bandit tasks. The meta-parameters are optimized over multiple parallel lifetimes and updated after the lifetimes have concluded, at which point the agent is reset and new lifetimes are sampled. The full details on the problem setting are given in appendix B.4. We run all experiments using JAX [Bradbury et al., 2018], evosax [Lange, 2022], and Tune [Liaw et al., 2018].

5.4.1 Sampling Correction and DiCE in Practice

We first investigate how the different meta-gradient estimators perform in an untruncated setting. In figure 5.1, we compare four different meta-gradient estimators in the bandit setting by training the learning rate parameters starting from nine initializations. The finite differences gradient estimator, which is unbiased, converges to the local optimum. The naive estimator does not use the sampling correction,

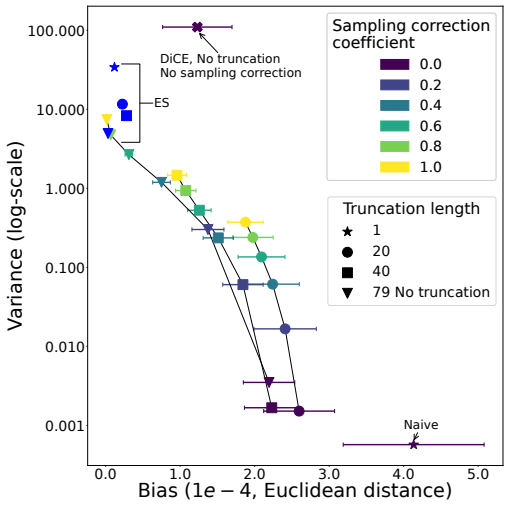


Figure 5.2: The bias and variance estimates for different meta-gradient estimators in the bandit setting. The error bars show the standard deviation of the bias when computed across bootstrap samples of the initializations. The different markers correspond to different truncation lengths as given in the legend. The different colors correspond to different sampling correction coefficients.

which leads to bias, causing the meta-parameters to move away from the local optimum. The sampling corrected gradient estimator converges to the same local optimum as the finite differences one, confirming that it is unbiased. We use DiCE [Foerster et al., 2018] to represent the category of meta-gradient estimators that use an estimate of the expected Hessian in the meta-gradient. Since the bandit problem does not have a time dimension, some of the methods developed for reducing variance of the expected Hessian estimator, including those proposed by Rothfuss et al. [2018] and Farquhar et al. [2019], simply reduce to DiCE. Using DiCE does not stop the meta-optimization from escaping the local optimum, demonstrating that the DiCE meta-gradient is biased.

5.4.2 Bias-Variance Tradeoff of Meta-Gradient Estimators

Next, we investigate the bias-variance tradeoffs of the meta-gradient estimators. We consider the tradeoffs with respect to the truncation length and the sampling correction coefficient ξ. We also show how estimators using DiCE and ES compare. The truncated meta-gradient estimators are computed by randomly sampling a fixed-length window of inner loop updates during the lifetime and restricting

backpropagation to that window. For each point in the comparison, we sample a large number of such windows. The objective of the agent is the same as before, i.e., the average lifetime return, which is approximated with the average return within the truncation window.

The results of the comparison are presented in figure 5.2. To estimate the bias, we compute the average Euclidean distance of the approximated true meta-gradient and the estimators at 25 points around the optimum of a bandit problem similar to the one shown in figure 5.1. To eliminate the effect of variance from the comparison, we compute high-precision approximations using a large number of samples. The true meta-gradient is approximated by finite differences. The error bars show the standard deviation of the bias estimated with 10k bootstrap samples from the 25 points, to reflect how much the bias varies across the meta-optimisation landscape. Because the sampling correction coefficient and length of the truncation window change the magnitude of the meta-gradient, while its direction may be more important in practice, we also present the same data using cosine similarity as the bias measurement in figure B.4; the results are qualitatively similar.

Effect of the Truncated Backpropagation Figure 5.2 shows that the general trend is for the bias to decrease with the increased truncation length. However, for multiple truncation lengths, the standard deviation of the bias of the uncorrected estimator overlaps with the mean of the other truncation lengths. This suggests that the bias of the uncorrected estimator does not decrease monotonically with the increased truncation length but may actually grow when a longer window is considered.

Effect of the Sampling Correction Coefficient Figure 5.2 confirms that the untruncated and sampling corrected meta-gradient estimator has no bias. It also shows that increasing ξ tends to decrease bias. However, we observe that for the truncated estimators, for some of the initializations the bias does not decrease with ξ. This is because the sampling correction terms themselves are biased in the truncated case and increasing their weight may increase the total bias at some parts of the meta-parameter space. We demonstrate this effect in

figure B.11 in the appendix, which shows a case where the bias increases with ξ for one initialization and decreases for another.

Variance of the Estimators The variance of the meta-gradient estimators grow rapidly with ξ and the truncation length. The variance of the unbiased estimator ($\xi = 1$, truncation length = 79) is multiple orders of magnitude higher than that of the naive estimator ($\xi = 0$, truncation length = 1). This is partly explained by the naive estimator having an order of magnitude smaller norm than the true gradient. In settings with long truncation lengths and large inner-loop batch sizes, the high variance from the sampling correction may prevent its use without more extreme variance reduction methods.

Evolution Strategies While ES has high variance, figure 5.2 shows a case where its variance is smaller than that of the unbiased backpropagation-based estimator. This is possible first because the black box design does not require an analogue to the sampling correction to account for the effect of the meta-parameters on the data distribution, and second because the variance of ES does not grow with the truncation length as rapidly as that of backpropagation based meta-gradients. In this case, the truncated ES estimators have larger variance than the untruncated one, because the truncated ones are averaged over random windows along the lifetime.

Pareto Frontier The bias-variance tradeoff presented in figure 5.2 gives rise to a Pareto frontier. Only two points are guaranteed to be on the frontier. The "naive" estimator with truncation length 1 is on the frontier because it has the lowest variance out of all of the estimators considered. The other point is the untruncated and sampling-corrected backpropagation-based estimator, which is the only point that is guaranteed to be unbiased. However, in practice, the bias from the smoothing of the objective in ES can be small and ES can therefore be on the frontier too depending on the specifics of the problem, especially the length of the lifetime. In general, ES has high variance, but its variance has a more favorable dependence on the length of the lifetime than the backpropagation-based estimators,

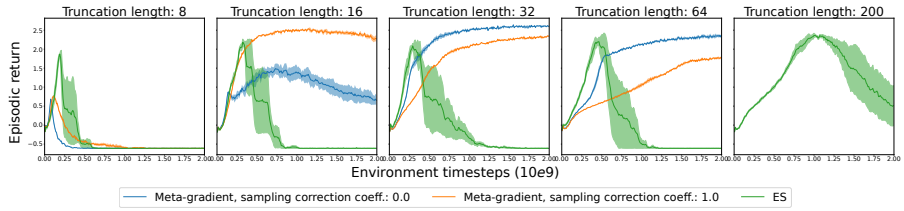


Figure 5.3: Meta-learning curves for the meta-gradient estimators in the MDP setting with truncated meta-optimization horizons. The curves show a smoothed version of the episodic returns throughout the meta-learning. The shading of the curves shows the standard error across seeds.

so for very long lifetimes it will have lower variance than the backpropagation-based counterpart. While the other estimators are not guaranteed to be on the frontier, in the problems we consider we see that, increasing the truncation length has a favorable bias-variance tradeoff to increasing sampling correction coefficient.

5.4.3 Using Sampling-Corrected Truncated Estimators

In the previous experiment, we found that the bias of the truncated meta-gradient estimator may increase with the sampling correction coefficient under some conditions. To investigate how using the sampling correction with the truncated estimator works in practice, we train the meta-parameters of the bandits with the different estimators. We train the meta-parameters for 200k steps using Adam [Kingma and Ba, 2014]. Figure 5.1 suggests that higher sampling correction values asymptote to better local optima. Thus, even though the sampling correction can increase the bias of truncated estimators, it can still lead to better meta-learning performance.

5.4.4 Sampling Corrections for the Full RL Problem

We now compare some of the meta-gradient estimators on or close to the Pareto frontier in the full RL problem. We consider a nonstationary gridworld environment adapted from Flennerhag et al. [2021] where the agent searches for one of two rewarding objects. The nonstationarity arises because the rewards of the objects are swapped every 6400 steps. Due to the nonstationarity, no unbiased gradient estimator exists. Nevertheless, we expect the truncation horizon and the sampling

correction to matter in this setting as well. The agent is implemented as a neural network without memory. The meta-learning problem is to learn a small neural network that outputs the entropy coefficient used for training the agent using a history of recent rewards as its input. The inner-loop trains the policy with a standard actor-critic algorithm and updates the parameters using SGD. The outer-loop updates the meta-parameters by gradient descent on the gradient given by equation (5.3), or ES gradient estimate on equation (2.16) and uses Adam for optimization. For full details on the setting and hyperparameters see appendix B.5.

The learning curves for training the entropy schedule with backpropagation-based and ES estimators are shown in figure 5.3. As with the bandits, the truncation length has a significant impact on the meta-optimization, largely determining what kind of local optimum it converges to. The sampling correction seems to lead to improved performance for the shorter truncation lengths, but results in very slow learning for the longer ones. ES learns quickly but diverges from the local optimum it reaches, making its overall performance poor. The meta-parametrization it learns is qualitatively different from those learned by the other estimators as can be seen from appendix B.5. These results suggest that at least in this setting, the truncation length is the main driver of the meta-learning performance, sampling correction can be important in some cases, but ultimately using it makes the meta-learning very slow, and that achieving stable meta-learning with ES can be difficult.

5.5 Related Work

In chapter 3, we provide a survey of literature relating to meta-gradient RL. In this section, we discuss and draw comparisons between the analysis presented in this chapter and the most closely related work. In concurrent works, Tang [2021] and Liu et al. [2021] also show that the DiCE-style Hessian estimator is a biased estimator of the true meta-gradient. In addition to discussing the bias from the DiCE-style meta-gradient estimation, we investigate the bias-variance tradeoff due to the sampling correction and truncation.

As confirmed by our experiments, long truncation lengths lead to high variance whereas short lengths increase bias. Bonnet et al. [2021] propose to make the bias-variance tradeoff in the truncation length by mixing multiple truncation lenghts in a single update. Flennerhag et al. [2021] take another approach to the tradeoff by proposing a new bootstrapping meta-objective. These works consider the bias-variance tradeoff due to truncated backpropagation, but they ignore the bias from missing the sampling correction terms, which we investigate together with the bias from truncation.

Besides RL, meta-gradients have been actively studied in supervised learning. Wu et al. [2018] show that for some problems the bias from truncating the optimization horizon yields suboptimal solutions. Metz et al. [2019] demonstrate how backpropagation through many update steps yields highly non-smooth objective surfaces making meta-learning challenging. Franceschi et al. [2017] show that for a small number of meta-parameters, forward-mode gradient computation can be used to overcome the memory constraint due to the long optimization horizon. These works demonstrate how the optimization horizon can be important for meta-learning and propose methods to alleviate the problems preventing meta-optimization over long truncation horizons. Despite their applications in supervised learning, these findings consider backpropagation through gradient-based updates and as such apply to meta-gradient estimation for RL as well.

5.6 Conclusion and Limitations

In this chapter, we investigated the estimation of meta-gradients in meta-RL. We showed that, contrary to claims in prior work, using an estimator of the expected Hessian in the meta-gradient estimator always adds bias, and is likely to also increase variance. Furthermore, we discussed the sampling correction needed for unbiased meta-gradient estimates, and described how it may be employed in the truncated optimization setting. In doing so, we explored the space of bias-variance tradeoffs that can be made in meta-gradient estimation. We also considered using ES instead of backpropagation for estimating meta-gradients. While ES can have lower variance

than backpropagation for long horizons, we were unable to achieve stable meta-learning using it. We found all of the estimators considered to have severe drawbacks: ES can be unstable, backpropagation scales poorly with the truncation length and is biased without sampling correction, which in turn results in even higher variance. Therefore, much work remains in developing better meta-gradient estimators that are capable of training complex meta-parametrizations on complicated problems.

The empirical analysis presented in this chapter is limited to only a small number of relatively simple problems, and is produced with finite resources for hyperparameter search. While we kept things simple in order to avoid introducing confounders, it is possible that the empirical results do not generalize to all other domains.

5.7 Statement of Authorship

Title of Paper: No DICE: An investigation of the bias-variance tradeoff in meta-gradients

Publication Status

- Published ✓
- Accepted for Publication
- Submitted for Publication
- Unpublished and unsubmitted work written in a manuscript style

Publication Details: Vuorio, Risto, Jacob Austin Beck, Gregory Farquhar, Jakob Nicolaus Foerster, and Shimon Whiteson. "No DICE: An investigation of the bias-variance tradeoff in meta-gradients." *In Deep RL Workshop NeurIPS 2021.* 2021.

Contribution to the paper: I was the lead author for the paper. I helped steer the overall research direction, implemented algorithms, conducted experiments, and helped write the manuscript.

Part II

Imitation Learning

6

Imitation Learning Literature Review

Contents

6.1 Introduction

In the previous chapters, we discussed the problem of meta-RL proposing new applications and analyses of meta-gradients. Now, we shift our focus into IL, which is the second focus area of this thesis. As discussed in chapter 1, IL studies learning behavior policies from expert demonstrations. When high-quality expert demonstrations are available, this can be a very efficient way of learning the imitator. However, acquiring a dataset of expert demonstrations for a particular problem of interest can be expensive as the expert is often a human that has to control the system manually or a policy trained using some more expensive algorithm than IL. Therefore, in order to reduce the cost of collecting demonstrations as

much as possible, we would like to be able to use demonstrations coming from experts that are not fine-tuned for the IL problem at hand. One compelling case where such data may be available is when we are training driving agents for traffic scenarios. Using the sensors of the self-driving car, we can collect data from a human expert demonstrating how to handle different traffic situations relatively cheaply. However, the human expert does not observe the world through the sensors of the car, but rather their own senses. The difference between the human senses and the car sensors may result in the human noticing and reacting to events and objects, such as sounds, smells, gestures, etc., in the traffic that are not observable to the self-driving car. This leads to the imitation gap problem we introduced in chapters 1 and 2. In chapter 2, we described some of the central algorithms used in IL and discussed the causal confounding and missing exploration information problems arising from the imitation gap. Next, we provide a broader literature review for IL and the imitation gap problem.

6.2 Literature Review

Mitigating the Distribution Shift The workhorse of IL is BC, which means just training a policy with supervised learning on the expert actions [Pomerleau, 1988]. BC suffers from a distribution shift problem, because the policy is trained on the state-action distribution defined by the expert policy but at test time, the actions are chosen by the policy itself. Even small imperfections in fitting the expert policy may result in problematic distribution shift as more actions are taken, due to a compounding of errors where the imitator deviates further and further from the expert data [Ross et al., 2011]. In chapter 2, we discussed a common mitigation to this issue called DAgger, which collects data using the imitator and relabels it with the expert actions [Ross et al., 2011]. If the expert's policy can be changed during data collection time, injecting noise to the expert actions may also help with the distribution shift, as it results in the expert visiting a larger set of states [Laskey et al., 2017]. Another potential fix to the problem is to make the policy predict multiple actions at a time, also known as action chunking, which

is a concept studied in psychology [Lai et al., 2022]. Action chunking has been found to be remarkably effective in helping with the compounding error resulting from the distribution shift [Bharadhwaj et al., 2024, Fu et al., 2024, Zhao et al., 2023]. Alternative training objectives such as diffusion [Chi et al., 2023], implicit BC [Florence et al., 2022], and auxiliary tasks prediction tasks [Rahmatizadeh et al., 2018] have been tried as well with strong results. Despite strong performance of the different objectives, using a standard BC objective with action chunking and other augmentations remains competitive for training policies for complex robotic applications [Fu et al., 2024, Zhao et al., 2023].

Trajectory-Conditional Policies Imitation learning in partially observable environments usually requires conditioning the action selection on more than just one observation from the environment. This is the case even when the expert and the imitator have access to the same information about the environment. Brohan et al. [2022], Shafiullah et al. [2022] consider using Transformers [Vaswani et al., 2017] as a sequence model to incorporate information throughout the history of the interaction. Other imitator architectures that can make use of historical information are RNNs [Mandlekar et al., 2021, Rahmatizadeh et al., 2018] and networks using temporal convolution [Jang et al., 2022]. Current state of the art for robotic applications makes use of Transformer-based policies that condition on the entire trajectory [Fu et al., 2024, Zhao et al., 2023].

Language-Conditional IL As LLMs [Achiam et al., 2023, Brown, 2020] have influenced nearly every aspect of AI, it is no surprise that they have also been applied to IL. One prolific area of research for using language models for imitation has been language-conditional IL, where the task the imitator is expected to solve is presented as a natural language description [Brohan et al., 2022, Jang et al., 2022, Shridhar et al., 2022, 2023]. A particular advantage of conditioning on language is the availability of strong pre-trained models that in combination with large IL datasets can enable generalization to new language defined tasks [Jang et al., 2022], semantic concepts [Shridhar et al., 2022], and robot morphologies [Brohan et al., 2022].

6.2.1 Inverse RL

As described in chapter 2, IRL is a closely related subject to IL, where instead of directly learning policies, the methods aim recover a reward function from expert demonstrations. A foundational work in IRL is maximum margin IRL [Abbeel and Ng, 2004], where the goal is to find a reward function under which the expert policy is optimal, while maintaining a large margin between the expert's trajectory and other suboptimal ones. This method paved the way for reward-based approaches to imitation learning, improving upon the limitations of direct behavioral cloning. An influential development following on maximum margin IRL is maximum entropy IRL [Ziebart et al., 2008a], which we discussed in more detail in section 2.3.1. Maximum entropy IRL uses the principle of maximum entropy to disambiguate among the possible reward functions. Besides learning reward functions, learning Q-functions is also possible and can lead to improved policy learning [Dvijotham and Todorov, 2010, Garg et al., 2021]. While IRL methods have advantages over BC, their RL component is typically not sample efficient because it has to spend time exploring policies that do not resemble the expert policy. Ren et al. [2024] proposes to update the policy on the expert data as well as its own exploration data to improve the sample efficiency.

Traditional IRL approaches are expensive to run due to the required RL component. However, in return, they can often deal better than BC with the distribution shift problem precisely because the RL algorithm learns to generalize over states via the online interactions. It may be an easier problem to learn a generalizable reward than a policy. This property of IRL has been put to good use in various applications of IRL, where they have been used for learning control policies for real-world platforms such as radio controlled helicopters [Ng et al., 2006], quadrupeds [Kolter et al., 2008, Ratliff et al., 2009], navigation policies in challenging terrains [Ratliff et al., 2009, Silver et al., 2010, Zucker et al., 2011], as well as route preferences similar to those of real taxi drivers [Ziebart et al., 2008b].

Another way of using RL to learn imitators based on the expert data is to consider a setting where a constant reward of +1 is assigned for taking the actions

the expert took and zero reward is assumed elsewhere [Reddy et al., 2019, Wang et al., 2019a]. A standard off-policy RL algorithm can then be used to learn a value function based on the data collected by the expert and the imitator. Swamy et al. [2021] develop the idea further and provides theoretical guarantees on the algorithm.

GAIL In chapter 2, we discussed GAIL [Ho and Ermon, 2016], which is an influential method in IL, closely related to the IRL setting. GAIL trains a discriminator to tell apart data coming from the expert and the imitator, and then uses the discriminator as reward for training the imitator with RL. GAIL can be used for learning policies in high-dimensional settings [Bronstein et al., 2022, Fu et al., 2018, Ho and Ermon, 2016] but has the drawback that the discriminator itself does not represent a meaningful reward function at convergence when the data distributions match. Adversarial inverse reinforcement learning [Fu et al., 2018, AIRL] is a further development of GAIL that provides a way to recover reward functions that explain the expert behavior. Additionally, Kolev et al. [2024] improve sample efficiency of GAIL by introducing a world model that is specialized to the IL setting.

6.2.2 Mitigations to the Imitation Gap

In this thesis, we focus on imitation gap problems arising from partial observability. Prior work addressing the imitation gap typically assumes privileged access to the true environment reward during training. Nguyen et al. [2022], Weihs et al. [2021] assume access to the true reward during training, and propose to bridge the imitation gap by training on a weighted imitation and RL loss. Other works propose to also integrate privileged information about the expert during training [Chen et al., 2019] in addition to the reward. For example, Elf Distillation [Walsman et al., 2023] studies an approach that mixes environment reward with online advice from the expert. Cai et al. [2021] propose several stages of training, including those on privileged expert states, to connect the imitator and expert's observation spaces. Separately, versions of the imitation gap have been considered by Kwon

et al. [2020], Ortega et al. [2021], Swamy et al. [2022b] that assume that no new exploratory behavior needs to be learned.

6.2.3 Transfer in IL

As discussed in chapter 2, the imitation gap resulting from partial observability, can be seen as an instance of a broader problem where the differences between the environments the expert and imitator are not limited to the information the agents receive. Other differences between the expert and imitator environments have been studied in the context of transfer learning in IL. Transferring policies across robot morphologies has been studied by Bousmalis et al. [2023], Fang et al. [2023], Padalkar et al. [2023], Team et al. [2024], Yang et al. [2023]. While in this case the state spaces of the environments are completely different, transferring across different robot morphologies could be seen as a problem that is solved by generalization to new states and does not necessarily require new algorithmic innovations relating to the imitation gap. A closely related problem is imitating humans from videos [Chen et al., 2021, Das et al., 2021, Edwards and Isbell, 2019, Nair et al., 2022, Radosavovic et al., 2023, Shao et al., 2021, Smith et al., 2019, Xiong et al., 2021]. While the data availability for human videos from online sources is often good, learning policies from it is challenging because the imitator necessarily does not share the state, observation, or action spaces with the demonstrator.

Another transfer setting in IL assumes that while actual expert data is expensive to collect, data from policies that are not necessarily experts for the current task but are from the same environment can be useful for learning the imitator [Cui et al., 2022, Lynch et al., 2020, Rosete-Beas et al., 2023, Wang et al., 2023]. Since the training data does not define the task unambiguously, this setting requires some other way of defining the goal, such as conditioning on a future state [Cui et al., 2022].

6.2.4 Meta-IL

The imitation gap problems assume a distribution of MDPs, making them related to the meta-RL [Beck et al., 2023] and meta-imitation learning (meta-IL) set-

tings [Dasari and Gupta, 2021, Duan et al., 2017, Englert and Toussaint, 2018, Finn et al., 2017b, James et al., 2018, Johns, 2021, Valassakis et al., 2022, Yu et al., 2018b]. In meta-IL, the aim is to train an imitation learning agent that can adapt to new demonstration data efficiently. Differently the imitation gap problems we consider, the tasks in meta-IL can vary in the reward function. For imitation learning to work with the new reward functions, demonstrations of policies maximizing the new rewards are required.

6.3 Conclusion

In this chapter, we provided a comprehensive review of imitation learning (IL), highlighting key approaches to mitigating the distribution shift in behavioral cloning, incorporating trajectory-based conditioning, and developing language-conditioned policies. We also examined IRL and its variants, such as GAIL, and discussed their unique advantages over direct policy imitation. Addressing the central theme of this thesis, we explored the concept of the imitation gap — particularly when it arises from partial observability — and surveyed methods designed to bridge this gap, including leveraging privileged information and reward-based training strategies. Furthermore, we touched upon transfer learning challenges in IL, such as transferring across different robot morphologies and imitating humans from videos, as well as the meta-imitation learning framework aimed at adapting quickly to new tasks.

Building upon these foundations, in chapter 7 we interpret the imitation gap as a causal confounding problem and proposes solutions using variational inference techniques. In chapter 8, we confront an even more challenging setting where the expert does not provide safe exploration guidance, necessitating more sophisticated approaches to learning from limited and potentially incomplete demonstrations.

7

Deconfounding Imitation Learning with Variational Inference

Contents

7.1 Introduction

As discussed in chapters 2 and 6, imitation gap refers to a set of problems in IL where the expert provides demonstrations in a different environment from the one the imitator faces. In the more severe instances of imitation gap, the expert data is not enough to learn the desired behavior in the first place. We will return to such problems in chapter 8. In this chapter, we focus on a more limited version of the imitation gap problem, where naïvely imitating an expert without accounting for the gap is not actively harmful, it just fails to achieve the task the expert demonstrates.

For an illustrative example, consider collecting data from a stochastic expert policy for IL in the classic game of Lunar Lander [Brockman et al., 2016] where the mapping from the actions to the thrusters on the lander is randomized. The expert occasionally employs the wrong thruster but, over the course of the whole episode, uses the thrusters in a pattern that results in a safe landing. In order to learn the correct mapping from the actions to the thrusters, the imitator has to account for the entire trajectory of interaction with the environment and choose the actions that most likely map to the correct thrusters. In such a situation, the imitator may take their own past actions as evidence for the values of the confounder. This can result in the imitator fixating on *the pattern* of actions the expert demonstrated instead of using its observations about the effects the actions have to identify the correct thrusters. This issue of *causal delusion* was first pointed out in Ortega and Braun [2010a,b] and studied in more depth by Ortega et al. [2021]. These delusions can also affect generative models including large language models [Ortega et al., 2021]. Instead of general generative modeling, we focus on a control setting where we have access to the true environment where the agent is going to be deployed.

Ortega et al. [2021] show that using the classic DAgger algorithm [Ross et al., 2011] solves this problem by querying the expert policy in the new situations the agent encounters to provide new supervision for learning the correct behavior. However, this kind of query access to the expert may not be available in practice. As another solution to the confounded imitation learning problem, Swamy et al. [2022b] propose to use IRL [Ng et al., 2000, Russell, 1998]. However, IRL typically

requires adversarial methods [Fu et al., 2018, Ho and Ermon, 2016], which are not as well behaved and scalable as supervised learning.

To get around the confounding problem without query access to the expert or IRL, we propose a practical algorithm based on variational inference. In our approach, an inference model for the latent variable is learned from exploration data collected using the imitator's policy. This inference model is used for inferring the latent variable on the expert trajectories for training the imitator and for inferring the latent variable online when the imitator is deployed in the environment. We show that, in theory, this algorithm converges to the expert's policy. In other words, we show that the expert's policy can be identified. Furthermore we show that under strong assumptions this identification can be carried out purely from offline data. Finally, we validate its performance empirically in a set of confounded imitation learning problems with high-dimensional observation and action spaces. These contributions can be summarized as follows:

- We propose a practical method based on variational inference to address confounded IL without expert queries. We verify its performance empirically in various confounded control problems, outperforming naïve BC and IRL.
- We propose a theoretical method for confounded IL, purely from offline data, without expert queries nor exploration.
- We provide theoretical insight into the proposed methods by proving under strong assumptions that the expert's policy can be identified.

7.2 Related Work

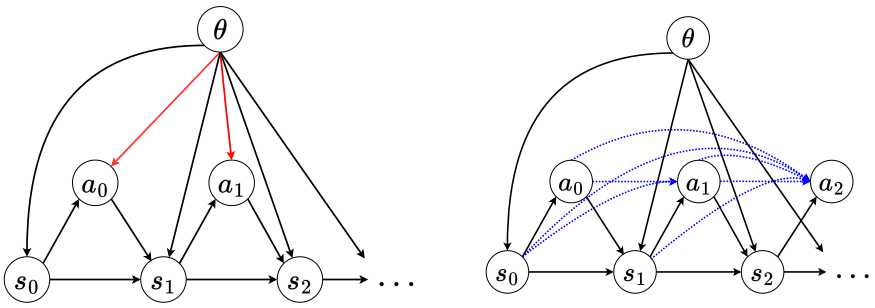
We provide a broad survey of literature on IL and the imitation gap problem in chapter 6. In this section, we focus on the immediately relevant related work for the problem considered in this chapter.

Ortega and Braun [2010a,b] and Ortega et al. [2021] pointed out the issue of latent confounding and causal delusions that we discuss in this chapter. In

particular, Ortega et al. [2021] propose a training algorithm that learns the correct interventional policy. Unlike our algorithm, their approach requires querying experts during the training. However, as we discuss in section 7.4, their solution has weaker assumptions and also applies to non-Markovian dynamics.

Most similar to our work is Swamy et al. [2022b], which finds theoretical bounds on the gap between expert behavior and an imitator in the confounded setting, when imitating via BC, DAgger [Ross et al., 2011], which requires expert queries, or inverse RL. Inverse RL suffers from two key challenges: (1) it requires reinforcement learning in an inner loop with a non-stationary reward, and (2) the reward is typically learned via potentially unstable adversarial methods [Fu et al., 2018, Ho and Ermon, 2016]. In contrast, our method trains the behavior policy using a well-behaved BC objective, which often enjoys better robustness and scalability compared to inverse RL.

There are various other works on the intersection of causality and IL which differ in setup. De Haan et al. [2019] consider the confusion of an imitator when the expert's decisions can be explained from causal and non-causal features of the state. It differs from our work as they assume the state to be fully observed, meaning it does not apply to our situation in which there is a latent confounder which needs to be inferred. This problem has been also discussed in Codevilla et al. [2019], Wen et al. [2020, 2022] and Spencer et al. [2021] under various names. Rezende et al. [2020] point out that the same problem appears in partial models that only use a subset of the state and find a minimal set of variables that avoid confounding. Swamy et al. [2022a] consider imitation learning with latent variables that affect the expert policy, but not the state dynamics, which is different from our case, in which the latent affects both the state and the expert's actions. Kumor et al. [2021] study the case in which a graphical model of the partially observed state is known and find which variables can be adjusted for so that conditional BC is optimal. An extension, Ruan et al. [2022], also handles sub-optimal experts.



(a) Expert (b) Ignorant policy

Figure 7.1: Bayes nets illustrating **(a)** an expert trajectory and **(b)** an imitator trajectory. The action chosen by the expert depends on the latent variable θ (red arrows) whereas the imitator's does not. Instead, the imitator conditions on the past states and actions (blue dotted arrows) to disambiguate the environment it is in.

7.3 Background

We begin by returning to the confounded imitation learning setting introduced in section 2.3.2. Following Ortega et al. [2021], we discuss how BC fails in the presence of latent confounders. We then define the interventional policy, the ideal (but a priori intractable) solution to the confounding problem.

7.3.1 Confounded Imitation Learning

In section 2.3.2, we extended the IL setting to allow for some latent variables $\theta \in \Theta$ that are observed by the expert, but not the imitator. Next, we discuss in more detail how this setting results in the problem of causal confouding in IL. We define the environment as a CMDP with latent space Θ distributed as $p(\theta)$. Here, we make the crucial assumption that the latent variable is constant in each trajectory. We assume there exists an expert policy $\pi_{\text{exp}}(a \mid s, \theta)$ for each θ. When it interacts with the environment, it generates the following distribution over trajectories τ:

$$p_{\text{exp}}(\tau \mid \theta) = p(s_0 \mid \theta) \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t, \theta) \pi_{\text{exp}}(a_t \mid s_t, \theta).$$

In this setting, the trajectories from the expert distributions are called confounded, because the states and actions have a common ancestor, the latent variable θ. The imitator does not observe this latent variable. It may thus need to implicitly infer it from the past transitions, so we take it to be a non-Markovian

policy $\pi_\eta(a_t \mid s_0, a_0, s_1, a_1, \ldots, s_t)$, parameterized by η. The imitator generates the following distribution over trajectories:

$$p_\eta(\tau \mid \theta) = p(s_0 \mid \theta) \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t, \theta)\ \pi_\eta(a_t \mid s_0, a_0, \ldots, s_t)\,.$$

The Bayesian networks associated with these distributions are shown in figure 7.1.

7.3.2 Naïve Behavioral Cloning

If we have access to a data set of expert demonstrations, we can use BC on the demonstrations to learn the maximum likelihood estimate of the expert's policy. In the confounded IL problem, at optimality, this learns the *conditional policy*:

$$\begin{aligned} \pi_{\text{cond}}(a_t \mid s_0, a_0, \ldots, s_t) &= \mathbb{E}_{\theta \sim p_{\text{cond}}(\theta \mid \tau)} \pi_{\text{exp}}(a_t \mid s_t, \theta)\,, \\ p_{\text{cond}}(\theta \mid \tau) &\propto p(\theta) \prod_t p(s_{t+1} \mid s_t, a_t, \theta) \pi_{\text{exp}}(a_t \mid s_t, \theta)\,, \end{aligned} \tag{7.1}$$

where $p_{\text{cond}}(\theta \mid \tau)$ is the posterior distribution over the latent variable given the trajectory likelihood under the expert's policy.

Following Ortega et al. [2021], consider the following example of a confounded multi-armed bandit with $\mathcal{A} = \Theta = \{1, \ldots, 5\}$ and $\mathcal{S} = \{0, 1\}$:

$$p(\theta) = \frac{1}{5},\ \pi_{\text{exp}}(a_t \mid s_t, \theta) = \begin{cases} \frac{6}{10} & \text{if } a_t = \theta \\ \frac{1}{10} & \text{if } a_t \neq \theta \end{cases},\ p(s_{t+1} = 1 \mid s_t, a_t, \theta) = \begin{cases} \frac{3}{4} & \text{if } a_t = \theta \\ \frac{1}{4} & \text{if } a_t \neq \theta. \end{cases} \tag{7.2}$$

The expert knows which bandit arm is special (and labeled by θ) and pulls it with high probability, while the imitating agent does not have access to this information. We define the reward in this bandit environment as $r_t = s_{t+1}$. However, note that we compare imitation learning algorithms that do not access the reward of the environment and only use the rewards for comparing the learned behaviors at evaluation time.

If we roll out the naïve BC policy in this environment, shown in figure 7.2, we see the causal delusion at work. At time t, inferring the latent by p_{cond} takes past actions as evidence for the latent variable. This makes sense on the expert demonstrations, as the expert knows the latent variable. However, during an imitator rollout, the

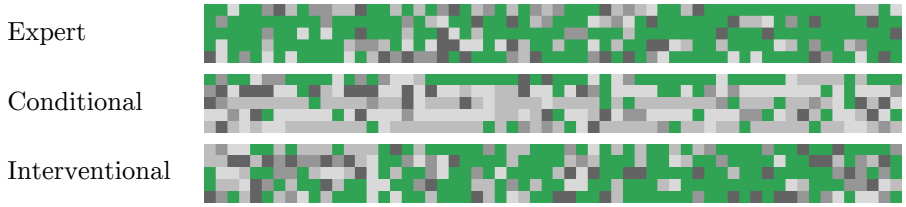


Figure 7.2: Actions from rollouts from bandit environment defined by equation (7.2). The x-axis is episode time. Five rollouts are shown vertically stacked for each of the expert, equation (7.1) and equation (7.3) policies. Colors denote actions, with the correct arm labelled green. As time progresses, the interventional policy tends to the expert's policy and chooses the correct arm more often than the others. In contrast, the conditional policy tends to repeat itself only converging on the expert policy by luck.

past actions are not evidence of the latent, as the imitator is blind to it. Concretely, the imitator will take its first action uniformly and later tends to repeat that action, as it mistakenly takes the first action to be evidence for the latent.

7.3.3 Interventional Policy

A solution to this issue is to only take as evidence the data that was actually informed by the latent, which are the transitions sampled from the dynamics distribution $p(s_{t+1} \mid s_t, a_t, \theta)$. This defines the following imitator policy:

$$\begin{aligned} \pi_{\text{int}}(a_t \mid s_0, a_0, \ldots, s_t) &= \mathbb{E}_{\theta \sim p_{\text{int}}(\theta|\tau)} \pi_{\text{exp}}(a_t \mid s_t, \theta)\,, \\ p_{\text{int}}(\theta \mid \tau) &\propto p(\theta) \prod_t p(s_{t+1} \mid s_t, a_t, \theta), \end{aligned} \tag{7.3}$$

where, in contrast to equation (7.1), the expectation is over a posterior defined only by the prior over the latent and dynamics distributions without a policy term. In a causal framework, this corresponds to treating the choice of past actions as interventions. To see this, consider the distribution $p(a_t, s_0, \ldots, s_t | do(a_0, \ldots, a_{t-1}))$. By classic rules of the do-calculus [Pearl, 2009], this is equal to

$$\int_\Theta p(\theta) p(a_t | s_t, \theta) \prod_{t'<t} p(s_{t'+1} | s_{t'}, a_{t'}, \theta) d\theta.$$

Taking the conditional $p(a_t | s_0, \ldots, s_t, do(a_0, \ldots, a_{t-1}))$ of that expression leads to equation equation (7.3). The policy given by equation (7.3) is therefore known as *interventional policy* [Ortega et al., 2021].

7.3.4 Failure of the Conditional Policy

Whether the conditional policy of equation (7.1) is able to imitate expert behaviors under latent confounding depends on the relative contribution of evidence due to the dynamics $p(s_{t+1} \mid s_t, a_t, \theta)$ and the policy $\pi_{\text{exp}}(a_t \mid s_t, \theta)$ when inferring the latent in the posterior $p_{\text{cond}}(\theta \mid \tau)$. If most evidence comes from the dynamics, we expect the conditional and the interventional policies to be similar. On the other hand, when most evidence comes from the policy decisions, we expect large delusion. This is for instance the case when the expert policy is deterministic and the dynamics stochastic, or when the latent influences the state transitions only in parts of the state space, but always affects the expert behavior. In appendix C.1, we quantitatively study the behavior of the conditional and interventional policies on the bandit example for varying amounts of dynamics and expert stochasticity.

7.3.5 Optimality of the Interventional Policy

To understand why the interventional policy may tend to the expert's policy consider $p_{int}(\theta|\tau)$ in equation (7.3). Applying it to the bandit example, each subsequent observation s_{t+1} after taking action a_t provides some evidence for the latent θ: if we find $s_{t+1} = 1$, then the latent θ is more likely to equal a_t, if we find $s_{t+1} = 0$, then θ is less likely to equal a_t. In figure 7.2, we see that this "true interventional" indeed approaches the expert's policy. This is further illustrated in section 7.6.1. In this case, the interventional policy thus presents a solution to the confounding problem. In the rest of this chapter, we focus on the question if and how it can be learned from data.

Note that the interventional policy may only achieve optimal performance after observing many time steps. However, it is not guaranteed to identify the latent as fast as possible. Faster identification may require choosing actions according to an active exploration strategy that uses its uncertainty about the latent to guide action selection [Zhou et al., 2019]. We leave the study of such exploration strategies for future work.

7.4 Deconfounding Imitation Learning

In this section, we discuss how, instead of the conditional policy, we can learn the interventional policy and thus fix the confounding problem. This improvement we call deconfounding the imitation learning. First, we illustrate how this is provably possible in theory under strong assumptions. We then show how to learn it in practice, even when these assumptions don't fully hold.

7.4.1 Identifying the Interventional Policy in Theory

Based on the graphical model shown in figure 7.1, and applying the rules of do-calculus, we can identify the interventional policy $\pi_{\text{int}}(a_t \mid s_0, a_0, \ldots, s_t) = p(a_t|s_0, \ldots, s_t, do(a_0, \ldots, a_{t-1}))$ by back-door adjustment [Pearl, 2009] if we observe the latent variable θ. This results in equation (7.3). But since the latent θ is unobserved, the interventional policy is not identifiable without further assumptions on the distribution.

However, we will now show that under some assumptions, the interventional policy does become identifiable. First, we assume that the latent variable θ of the MDP is static, or in other words stays fixed during the interaction of the policy with the MDP. This makes the dynamics and expert policy stationary in each trajectory. If we combine this with the strong assumption of *recurrence*, commonly used in theoretical reinforcement learning [Watkins and Dayan, 1992, Thm. 1], the interventional policy becomes identifiable.

Assumption 1. *We assume that the MDP is* recurrent *[Norris, 1997, Sec. 1.5], meaning that all state-action pairs s, a are reached infinitely often in each trajectory.*

If the MDP is recurrent with the expert policy, it is possible to identify the interventional policy in theory from an infinite dataset $\mathcal{D}$ of expert demonstrations of infinite length. On each individual trajectory τ_i with index i, we are able to count the state and action transitions to estimate the transition probability $\hat{p}_i(s'|s, a)$ and expert policy $\hat{\pi}_i(a|s)$ of that trajectory. We can then compute the likelihood of a dataset trajectory i given a state-action sequence $(s_0, a_0, \ldots, s_t)$ based on the

estimated environment transitions, $\prod_t \hat{p}_i(s_{t+1}|s_t, a_t)$. The corresponding belief over dataset trajectories i follows from Bayes' theorem as

$$\hat{p}_{\text{int}}(i|s_0, a_t, ..., s_t) = \frac{\prod_t \hat{p}_i(s_{t+1}|s_t, a_t)}{\sum_{i'} \prod_t \hat{p}_{i'}(s_{t+1}|s_t, a_t)} .$$

Then we can estimate

$$\hat{\pi}_{\text{int}}(a_t|s_0, a_0, .., s_t) = \mathbb{E}_{i\sim\hat{p}_{\text{int}}(i|s_0,a_t,...,s_t)}\hat{\pi}_i(a_t|s_t) . \tag{7.4}$$

This successfully identifies the interventional policy:

Theorem 1. *From infinitely many demonstrations of infinite length from a MDP that is recurrent with the expert policy π_{exp}, we have that $\hat{\pi}_{\text{int}}$ from equation (7.4) equals π_{int} from equation (7.3).*

Proof. Because of the recurrence assumption, we can correctly estimate $\hat{p}_i(s'|s, a) = p(s'|s, a, \theta_i)$ and $\hat{\pi}_i(a|s) = \pi_{\text{exp}}(a|s, \theta_i)$. Then, suppressing a normalization factor that is constant in a_T, we write:

$$\begin{aligned}
\pi_{\text{int}}(a_T|s_0, ..., s_T) &\propto \int_\Theta p(\theta) \left[\prod_t p(s_{t+1}|s_t, a_t, \theta)\right] \pi_{\text{exp}}(a_T|s_T, \theta) d\theta \\
&\propto \int_\Theta \sum_i p(\theta|i)p(i) \left[\prod_t p(s_{t+1}|s_t, a_t, \theta)\right] \pi_{\text{exp}}(a_T|s_T, \theta) d\theta \\
&\propto \sum_i p(i) \left[\prod_t p(s_{t+1}|s_t, a_t, \theta_i)\right] \pi_{\text{exp}}(a_T|s_T, \theta_i) \\
&\propto \sum_i p(i) \left[\prod_t \hat{p}_i(s_{t+1}|s_t, a_t)\right] \hat{\pi}_i(a_T|s_T) \\
&\propto \hat{\pi}_{\text{int}}(a_T|s_0, ..., s_T)
\end{aligned}$$

where we recognized $p(\theta|i)$ as the (Dirac) delta distribution around θ_i, and $p(i)$ is the uniform distribution over samples from the dataset $\mathcal{D}$. □

In this proof, the stationarity of the dynamics and expert make it possible to collect statistics across the time steps into distributions. Furthermore, the proof crucially relies on the recurrence assumption, as it allows us to evaluate the likelihood of any state-action sequence on each of the trajectories' distributions.

Without this assumption, we will in general only be able to learn an approximation of the likelihood from state-action pairs observed in the trajectory.

The interventional policy estimated in equation (7.4) requires marginalization over the entire demonstration dataset and thus isn't very practical. Instead, we propose to match the expert data with a learned latent variable model $p_{\psi,\eta}$

$$p_{\psi,\eta}(\tau) = \int_{\hat{\Theta}} p_\psi(\hat{\theta}) \prod_t p_\psi(s_{t+1}|s_t, a_t, \hat{\theta}) \pi_\eta(a_t|s_t, \hat{\theta}) d\hat{\theta}. \tag{7.5}$$

Similar to the argument in theorem 1, under the same assumptions, if we match the ground-truth demonstration likelihood $p(\tau)$ with the latent variable model $p_{\psi,\eta}$, then the interventional policy derived from the latent variable model $p_{\psi,\eta}$ matches the ground truth interventional policy π_{int}.

7.4.2 Learning from Demonstrations and Explorations

If each expert demonstration covers the entire state-action space, we have shown that we can estimate state dynamics and expert policy, and thus the interventional policy using equation (7.4) or the latent variable model in equation (7.5).

What if this strong assumption on the expert data does not hold? We can make the problem easier by allowing our agent to interact with the environment during training. That allows us to see more samples of the environment dynamics, making up for the lack of recurrence guarantees. Allowing sampling new training data from the environment is a common modification to the imitation learning problem, used, for example, in IRL. The distribution over the trajectories in the new data is given by

$$p_{\text{expl}}(\tau) = \int_{\Theta} p(\theta) \prod_t \pi_{\text{expl}}(a_t|s_t) p(s_{t+1}|s_t, a_t, \theta) d\theta \,, \tag{7.6}$$

where $\pi_{\text{expl}}(a|s)$ is the exploration policy we use for collecting the data. The corresponding latent model is given by

$$p_{\psi,\text{expl}}(\tau) = \int_{\hat{\Theta}} p_\psi(\hat{\theta}) \prod_t \pi_{\text{expl}}(a_t|s_t) p_\psi(s_{t+1}|s_t, a_t, \hat{\theta}) d\hat{\theta} \,, \tag{7.7}$$

where we assume that we can evaluate the likelihood of the exploration policy and approximate the dynamics distribution by the latent-variable model p_ψ.

We propose to represent the latent model of equation (7.7) as a variational autoencoder (VAE) [Kingma and Welling, 2014], using an inference model $q_\phi(\hat{\theta}|\tau)$ that infers the latent variable from a trajectory. Training the VAE involves maximizing the evidence lower bound (ELBO), a lower bound of the log likelihood, with respect to ψ and ϕ:

$$\begin{aligned}\mathbb{E}_{\tau\sim p_{\text{expl}}} \log p_{\psi,\text{expl}}(\tau) \geq \mathbb{E}_{\tau\sim p_{\text{expl}}}\mathbb{E}_{\hat{\theta}\sim q_\phi(\hat{\theta}|\tau)}\Bigg[&\log p_\psi(\hat{\theta}) - \log q_\phi(\hat{\theta}|\tau) \\ &+ \sum_t \log \pi_{\text{expl}}(a_t|s_t) + \log p_\psi(s_{t+1}|s_t, a_t, \hat{\theta})\Bigg] \\ = \mathbb{E}_{\tau\sim p_{\text{expl}}}\mathbb{E}_{\hat{\theta}\sim q_\phi(\hat{\theta}|\tau)}\Bigg[&\log p_\psi(\hat{\theta}) - \log q_\phi(\hat{\theta}|\tau) \\ &+ \sum_t \log p_\psi(s_{t+1}|s_t, a_t, \hat{\theta})\Bigg] + \text{const} = \mathcal{L}_{\text{VI}} \end{aligned} \tag{7.8}$$

where we can ignore the exploration policy likelihood as it does not depend on the parameters. At optimality, the inference model learns the model posterior given by

$$q_\phi(\hat{\theta}|\tau) = p_{\psi,\text{expl}}(\hat{\theta}|\tau) \propto p_\psi(\hat{\theta}) \prod_t p_\psi(s_{t+1}|s_t, a_t, \hat{\theta}),$$

assuming the model is sufficiently expressive to represent the posterior. Note that to be able to train the inference model as the encoder in a VAE, we condition it on the entire trajectories, which makes it incompatible with online use when sampling trajectories from the environment. We revisit this assumption in the next section.

Crucially, the learned inference model $q_\phi(\hat{\theta}|\tau)$ does not depend on the likelihood of the policy selecting the actions. This is because we explore with a policy that—unlike the expert—does not depend on the latent θ. Therefore, we learn exactly the interventional latent inference model from equation (7.3), with the learned dynamics.

In the next step, we use the inference model $q_\phi(\hat{\theta}|\tau)$ together with a learned dynamics model $p_\psi(s_{t+1}|s_t, a_t, \hat{\theta})$ to learn an imitation policy $\pi_\eta(a|s, \hat{\theta})$, that approximates the expert policy $\pi_{\text{exp}}(a|s, \theta)$. Again, we use variational inference and train the model $p_{\psi,\eta}$ from equation (7.5) with samples from the demonstrations. Thus, we maximize η in the ELBO

$$\mathbb{E}_{\tau\sim p} \log p_{\psi,\eta}(\tau) \geq \mathbb{E}_{\tau\sim p}\mathbb{E}_{\hat{\theta}\sim q_\phi(\hat{\theta}|\tau)}\left[\log \pi_\eta(a_t|s_t, \hat{\theta})\right] + \text{const} = \mathcal{L}_{BC}\,, \tag{7.9}$$

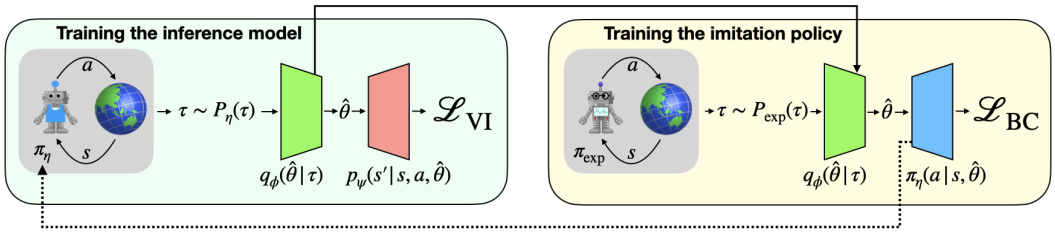


Figure 7.3: Overview of the method. On the left, the inference model consisting of the encoder q_ϕ and decoder p_ψ is trained using variational inference. Training data is sampled from the environment using an exploratory policy. The dotted arrow depicts how the exploratory policy can optionally be the imitation policy π_η trained on the right. On the right, the imitation policy is trained with behavioral cloning on expert data. The expert does not need to interact with the environment at training time. Instead, stored expert trajectories can be used. To learn the deconfounded policy, the encoder trained on the left is used for inferring the latent variable on the expert trajectories.

which effectively becomes a behavioral cloning loss. Here we omitted terms constant in the policy parameters η.

The training based on these two objectives is illustrated in figure 7.3. By optimizing the loss in equation (7.8), we learn a model to infer the latent variable just based on the dynamics of a trajectory; by optimizing the loss in equation (7.9), we learn a policy conditional on that latent. Combined, we learn to approximate the interventional policy of equation (7.3).

7.5 Practical Algorithm

We now present a practical algorithm for training an agent from expert data in the presence of latent confounders. As outlined in section 7.4, we learn to infer the latent variable by interacting with the environment using an exploratory policy, and learn to clone the expert on the demonstrations conditioned on the inferred latent. At test time, the agent alternates between updating its belief about the latent variable and imitating an expert dictated by its current belief. In appendix C.3, we describe an algorithm that does not require the ability to gather more data interactively, but faces a more difficult learning problem in practice.

Components The agent consists of: 1) an inference model q_ϕ, which maps trajectories $\tau = (s_0, a_0, \ldots, s_n)$ to a belief over a latent variable $q_\phi(\hat{\theta} \mid \tau)$; 2) a

dynamics model p_ψ mapping latent, state, and action to a distribution over the next state $p_\psi(s' \mid s, a, \hat{\theta})$; and 3) a latent-conditional policy $\pi_\eta(a \mid s, \hat{\theta})$. An overview of the training algorithm is presented in figure 7.3.

Training the Model for Online Inference The inference model training closely follows the outline given in section 7.4. However, since at test time, the agent needs to infer the latent online from partial trajectories, we adjust the model and the ELBO of equation (7.8) slightly. At timestep t, the encoder q_ϕ takes as input the trajectory observed until timestep t, and predicts a distribution of the belief over the latent $\hat{\theta}$.

We follow Zintgraf et al. [2020] by defining the prior at timestep t as the inferred latent distribution from timestep $t-1$, starting with a diagonal Gaussian with unit variance as the initial prior. This process is similar to Bayesian filtering, where beliefs about the state of a process are updated sequentially in response to new evidence. As such, each prior evolves based on the latest observed data, similar to the updating mechanisms in Kalman filters and other Bayesian state estimators [Kalman, 1960, Krishnan et al., 2015]. The modified ELBO can then be written as

$$\hat{\mathcal{L}}_{VI} = \mathbb{E}_{\tau \sim p_{\text{expl}}}\Bigg[\mathbb{E}_{\hat{\theta} \sim q_\phi(\hat{\theta} \mid \tau_{:t})}\Big[\log p_\psi(s_{t+1} \mid s_t, a_t, \hat{\theta})\Big] - \beta D_{KL}\Big(q_\phi(\hat{\theta} \mid \tau_{:t}) \,\Big\|\, q_\phi(\hat{\theta} \mid \tau_{:t-1})\Big)\Bigg], \tag{7.10}$$

where D_{KL} is the KL divergence and $\tau_{:t}$ denotes the trajectory until timestep t. Following prior work on VAEs, we use a coefficient β for the prior regularization [Higgins et al., 2017]. Note that while the objective in equation (7.10) remains similar to the VAE objective in equation (7.8), it no longer represents an autoencoding objective, since future states are decoded given the inference conditioned only on the past.

As an exploration policy, we use the latent-conditional policy π_η conditioned on $\hat{\theta}$ inferred by q_ϕ. This is a convenient choice because using π_η means we do not have to train multiple policies. Furthermore, using π_η for exploration biases the training data distribution for the inference model toward data that the policy encounters when it is deployed in the environment potentially improving the generalization

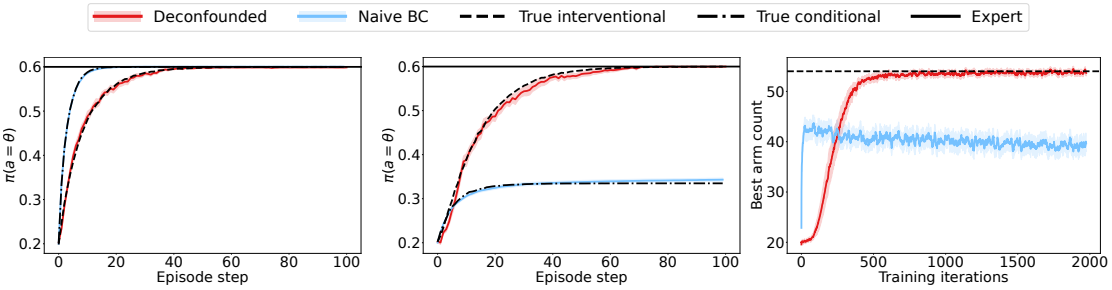
of the inference model. In practice, we condition the policy on the mean of the inferred distribution instead of a sample from it. We find that using either does not make a big difference. Other exploration policies may be used as long as they explore sufficiently diverse trajectories and do not depend on the true latent θ.

As described in section 7.4, the learned inference model is used for inferring the latents on the expert trajectories τ_e^j. The policy π_η is then trained to minimize equation (7.9). We show the pseudocode for the full training algorithm in appendix C.2.

Test Time At test time, the agent faces an environment with an unknown latent and needs to adapt to the correct expert behavior. We solve this problem by alternating between updating a posterior belief over the latent and acting under the current belief. Concretely, the agent initially samples a latent from the prior $\hat{\theta} \sim \mathcal{N}(0, \mathbb{1})$ and an action $a \sim \pi_\eta(a|s, \hat{\theta})$ to imitate the expert corresponding to that latent. It observes the state transition and computes the posterior belief with the inference network. Another latent is sampled from the updated belief, and so on. In practice, as during exploration, we do not sample from the inferred distribution but instead condition the policy on the mean. Once the inference has converged to match the true latent for the environment, the true expert for the environment will be imitated consistently. We summarize the test-time behavior in pseudocode in appendix C.2.

7.6 Experiments

To test our method in practice, we conduct experiments in the multi-armed bandit problem from Ortega et al. [2021] and in multiple control environments. The implementation of the experiments can be found on github. We aim to answer three questions: 1) How big is the effect of confounding on naïve BC — large enough to justify the use of specialized methods? 2) Is our algorithm capable of identifying the interventional policy? 3) How well does the interventional policy imitate the expert?



(a) Best arm probability on expert data (b) Best arm probability on online data (c) Learning curves

Figure 7.4: Imitation learning in a multi-armed bandit problem. The shading shows the standard error of the mean. The left panel compares the policies when evaluated on trajectories sampled by the expert policy. The x-axis is the step on the trajectory and the y-axis is the probability of choosing the best arm. The middle panel is otherwise the same, except run on trajectories sampled online with the policies themselves. The right panel shows the learning curves. The x-axis shows training iterations and the y-axis shows the number of times the best arm is chosen by the policy under training during a trajectory with 100 time steps. The curves are averages for sliding window of length 10 training iterations.

7.6.1 Investigating Deconfounding in a Multi-Armed Bandit

We begin the empirical study by experimenting with the multi-armed bandit problem proposed by Ortega et al. [2021] and described in section 7.3. The expert policy is defined in equation (7.2). We consider episodes of length 100. As we are only interested in the effects of confounding on imitation learning, and not in effects arising from over-fitting a small dataset, we generate new training data from the expert for each update of the learning algorithms. Each learning algorithm is run for ten independent seeds and the results are averaged. The hyperparameters for the algorithms are provided in appendix C.4.

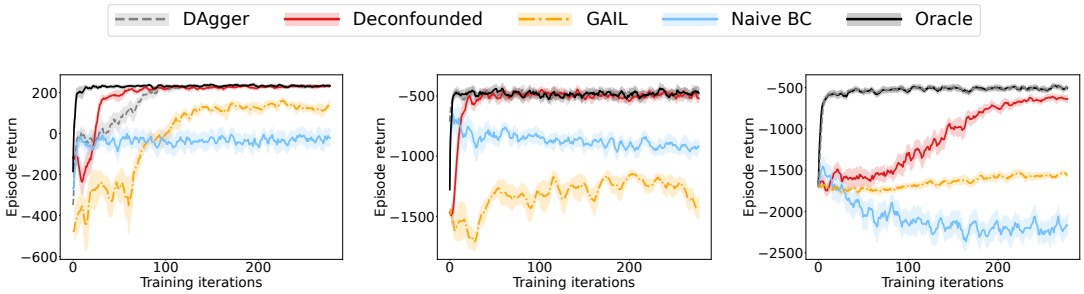
Naïve BC and the Conditional Policy To answer our first question, we compare a naïve BC to the true conditional policy described in section 7.3. As the latter policy is non-Markovian, we also allow the BC policy to observe the history. One way to enable this adaptation is to equip the agent with a memory, which the agent can learn to use for representing its belief about the latent variable. To provide such a memory, we implement the imitation learner as a recurrent neural network (RNN). figure 7.4 a) shows the probability the different policies assign

to choosing the best arm when evaluated on data collected by the expert policy. We see that the naïve BC agent learns a policy that matches the true conditional policy closely. This results in problems for the policy learned with naïve BC when it is deployed in the environment and has to choose the actions itself, as shown in figure 7.4 b). The naïve BC closely tracks the action probability of the true conditional policy, which performs much worse than the expert policy. These results suggests that it has suffered the full impact of the confounding problem.

Deconfounded Imitation Learning and the Interventional Policy To answer our second question, we implement the deconfounded imitation learner as described in section 7.5. Figure 7.4 a) and b) show that the proposed method closely matches the true interventional policy both on expert trajectories and online. Figure 7.4 c) shows the number of times the policies chose the best arm during an episode. Our deconfounded imitator converges to the true interventional policy, showing that it learned to optimally imitate the expert in the presence of latent confounders, and answering our third question. This near-perfect imitation performance comes at the price of requiring exploration data to train the inference model as well as an increased number of training iterations needed for convergence.

7.6.2 Demonstrating Deconfounding in Confounded MDPs

Next, we demonstrate that the confounding issue affects MDPs with considerably more complex dynamics than the bandit and that the answers to the three questions do not change with the increased complexity. For `LunarLander-v2` [Brockman et al., 2016], we consider a modified version with unknown key bindings: a latent θ specifies a permutation in the map between two of the the agent's actions and the behaviors (firing the left and right engines of the space craft). This permutation is known to the expert, but not the imitator. For `HalfCheetahBulletEnv-v0` [Coumans and Bai, 2016–2021], we modify the environment similarly to Swamy et al. [2022b] by varying the target speed. The expert observes the true target speed while the imitator only observes a noisy indicator, which shows whether it is running faster



(a) Lunar Lander (b) Half Cheetah (c) Ant

Figure 7.5: Experiments in our confounded, stochastic environments. We show the episodic return of each agent over the course of training. The curves for the are averages sliding window of length 5. The shading shows the standard error of the mean.

or slower than the target speed. For `AntGoal-v0` [Todorov et al., 2012], we consider a version, where the task is to run to a goal randomly sampled from within a circle around the starting point. The expert observes the true goal coordinates, while the imitator only observes a noisy indicator of the goal direction and distance. In all environments, in addition to providing the imitator with less information about the latent variable than the expert, we make the environment somewhat stochastic. These changes make the naïve imitators try to infer the latent from the more deterministic expert actions rather than the stochastic dynamics, which results in the confounding problem. For full details about the environments, and how the confounding problem may arise in each of them, please see the appendix C.5.

The expert policies are trained with proximal policy optimization [Schulman et al., 2017]. In order to avoid finite-sample-size effects, we use an infinite-size training dataset by generating expert trajectories on the fly. Each learning algorithm is run for 5 or 6 independent seeds and the results are averaged. Additional details are provided in appendix C.5.

Naïve BC Like in the bandit example, we first analyze how strongly confounding affects the naïve BC policy. The naïve learner is again implemented as an RNN. In figure 7.5 we see that the naïve BC agent fails to imitate the expert behavior and performs much worse than an oracle imitator, which is otherwise exactly the same setting but it is trained with knowledge of the latents. This failure to generalize

is again evidence for causal delusions: the agent learned to infer the latent from the expert behavior rather than the noisy dynamics of the environment.

Deconfounded Imitation Learning To solve the confounding issue in these environments, we first test the DAgger algorithm [Ross et al., 2011], which queries the experts during training. We find that it indeed solves the confounding problem: the agent quickly approaches the performance of the oracle imitator in all three environments.

The DAgger performance serves as an unachievable upper bound for our method, as it not only solves the confounding issue but also reduces the more common distribution shift issue present in imitation learning. However, recall that the expert policy may be defined by, e.g., a human expert, who would be expensive or impossible to query at imitation learning time making DAgger a potentially difficult method to use in practice.

Can our deconfounded imitator also solve the confounding problem? We test the deconfounded imitation learning algorithm described in section 7.5 on the control environments. We find it beneficial to modify the described algorithm in two ways: 1) extending the reconstruction loss in equation (7.10) to multi-step predictions [Hafner et al., 2020]; 2) conditioning the policy on the inferred latent $\hat{\theta}_t^j$ at the current timestep t instead of the last time step $\hat{\theta}_H^j$ when training the policy. See appendix C.5 for details.

In figure 7.5, we see that the deconfounded learner clearly outperforms the naïve BC baseline, matching the performance of the policy learned by DAgger in two out of three environments and coming close in the third. While it is hard to interpret what functions high-dimensional neural networks have learned exactly, matching the performance of DAgger, which we know can recover the interventional policy, is encouraging. It suggests that the deconfounded agent learned to infer the latent from environment transitions and learn a policy that acts like the expert under the inference. In the Ant environment, it is impossible to infer the direction from the noisy indicator from a single observation. Therefore, the agent may start moving

the wrong way in the beginning of the episode. Recovering from such a false start may be much easier with the help of extra supervision from the expert, giving DAgger a particularly strong advantage in this environment.

Inverse Reinforcement Learning In theory, IRL is capable of recovering the expert performance even in confounded environments [Swamy et al., 2022b]. We test this theory in practice, by comparing to generative adversarial imitation learning (GAIL) [Ho and Ermon, 2016]. We implemented GAIL closely following a popular publicly available implementation[1] and using recurrent PPO by Raffin et al. [2021] as the RL algorithm. To stabilize GAIL, we sample four times more data for every training step compared to the other algorithms.

In LunarLander, GAIL is able to achieve higher returns than naïve BC, but does not recover the performance of the proposed method. In HalfCheetah and Ant, GAIL does not recover the performance of naïve BC. In appendix C.4, we provide results for GAIL in the bandit environment from section 7.6.1. We found that in the bandit, GAIL does not recover the interventional policy and therefore does not converge to the expert behavior during an episode. This is because the reward function defined by GAIL can always be maximized by a deterministic policy, even in the case when the expert is a stochastic policy, like in the bandit. From these results, we conclude that while IRL is in theory capable of solving confounded imitation learning problems, where the expert is a deterministic policy, getting it to work well can be difficult in practice.

7.7 Limitations

While the proposed method recovers the policy DAgger learns under ideal conditions, it does not address the original challenge DAgger is designed to solve. That is, if the expert data does not cover the state-action space sufficiently, the proposed method may not be enough to learn the optimal policy. Furthermore, while the proposed method does not require access to the expert, it does require sampling

[1] `https://github.com/HumanCompatibleAI/imitation`

access to the environment similarly to IRL. To find the deconfounded policy fully offline, an algorithm outlined in the in section 7.4 could be used. For the methods to work provably, we need to make strong assumptions of recurrence.

Following Ortega et al. [2021] we model the confounded IL setting with a CMDP where the latent stays fixed during the entire interaction of the policy with the environment. This may be a limiting assumption, for example in a driving scenario, where the environment may have latent factors that govern the dynamics over a small section of the road but the agent is expected to be able to traverse many different sections.

Finally, while the environments we experiment in are commonly used to evaluate RL and IL policies, we acknowledge that these are fairly limited domains and do not reflect the full complexity of training useful policies for the real world.

The components of our algorithm, variational inference and behavioral cloning, are commonly used techniques elsewhere in machine learning. Therefore, scaling the proposed method to real world problems should be possible. However, we did not optimize for sample complexity of either learning algorithm, which would limit the application of our method to real world problems. Developing a more sample efficient version of the algorithm and testing it in real world domains is an exciting avenue of future work.

7.8 Conclusion

Naïve imitation learning algorithms can fail in the presence of latent confounders — for instance when the expert has access to more information than the imitator. This work presents a breakdown of this confounding problem. First, we studied under which conditions latent confounding impacts the performance of BC. We demonstrated that this issue is more relevant when there is substantial stochasticity in environment transitions or the expert policy is nearly deterministic. We then analyzed in which settings the interventional policy, which solves the confounding issue, is identifiable without query access to the expert.

Informed by the theoretical results, we proposed a practical algorithm for deconfounding imitation learning with variational inference that provably converges to the interventional policy. While this problem has been previously studied in theory with inverse RL, we propose a novel variational inference solution, which we analyze theoretically and evaluate in practice. Finally, we evaluated the proposed method with experiments in a multi-armed bandit and confounded MDPs. We found it was able to learn interventional policies in all of the settings, alleviating the confounding problem that limits naïve imitation learning.

In the next chapter, we tackle the more difficult imitation gap setting where safe exploration is not possible by sampling expert policies.

7.9 Statement of Authorship

Title of Paper: Deconfounding Imitation Learning with Variational Inference

Publication Status

- Published ✓
- Accepted for Publication
- Submitted for Publication
- Unpublished and unsubmitted work written in a manuscript style

Publication Details: Vuorio, Risto, Pim De Haan, Johann Brehmer, Hanno Ackermann, Daniel Dijkman, and Taco Cohen. "Deconfounding Imitation Learning with Variational Inference." *Transactions on Machine Learning Research*. 2024.

Contribution to the paper: I was the co-lead author for the paper. I helped steer the overall research direction, implemented algorithms, conducted experiments, and helped write the manuscript.

8

A Bayesian Solution To The Imitation Gap

Contents

8.1 Introduction

In the previous chapter, we considered an imitation gap problem, where naïvely imitating an expert does not result in a catastrophic failure and proposed a method for learning imitators in that setting. However, in practice, this may often be a

too strong assumption to make. For example, consider training a robotic agent for navigating an environment with steep ledges from which the robot may fall off. The expert knows the location of the ledges and steers well clear of them. Navigating the environment without precise knowledge of the locations of the ledges may require behavior that differs considerably from the expert behavior. This safe behavior could include moving more slowly and turning around when a ledge is detected. This exploratory behavior is never demonstrated, leading to an imitation gap, and hence naïvely imitating expert demonstrations leads to suboptimal behavior that is potentially catastrophic.

At first glance, this problem seems intractable, since we cannot imitate behavior that is not demonstrated. Indeed, many prior solutions to the imitation gap often require privileged access to online reward information [Nguyen et al., 2022, Weihs et al., 2021]. The key insight in this chapter is that the remaining uncertainty may be characterized in the form of a prior, specified over the cost of exploration in unobserved states. By cost of exploration, we mean the relative cost of how desirable it is for the agent to visit a state an expert would never visit compared to sticking exclusively to the states the expert explores. This leads us to propose a fully **B**ayesian solution to the **I**mitation **G**ap (BIG) which learns to behave Bayes-optimally at test-time, although its value will naturally be a function of the agent's uncertainty.

We demonstrate how the prior can integrate several sources of information available before test time: we specify an initial reward prior, from which we may infer a posterior given a set of expert demonstrations and simulator, using Bayesian IRL [Ramachandran and Amir, 2007, BIRL] with successor features [Barreto et al., 2017, Brown and Niekum, 2019, Filos et al., 2021]. We also specify a reward prior over key exploration states to specify uncertainty about the *cost of exploration* where there is an imitation gap. This allows BIG to optimally trade off any remaining uncertainty about the true environment state using a Bayes-optimal policy, with the predictive reward under these priors as the fixed belief over rewards at test time. When there is an imitation gap, the reward prior can influence the agent's behavior instead, yielding policies that *balance exploration to reduce the uncertainty*

in the environment with exploitation of expert demonstrations. When there is no imitation gap, the agent can directly imitate expert behavior.

We evaluate BIG across a number of imitation gap problems, and show that it can scale to environments with pixel-based observations. In each case, we can recover suitable reward functions from *only expert demonstrations and the cost of exploration prior*. Our contributions can be summarized as follows

- We propose an algorithm BIG to the imitation gap problem, which learns to behave Bayes-optimally at test time.
- We demonstrate that the Bayesian prior can integrate the additional information required for IL in the imitation gap setting from multiple sources.
- We evaluate the BIG in multiple imitation gap problems demonstrating that it learns the Bayes-optimal policy.

8.2 Problem Setting

We consider the imitation gap problem setting introduced in section 2.3.2, where the environment is modeled as a CMDP with a context variable $\theta \in \Theta$. We are interested in policies that optimize the contextual expected, discounted return $\mathbb{E}_{p^{\pi}_{\infty}(\theta)}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$ where $p^{\pi}_{\infty}(\theta)$ is the distribution over infinite-horizon trajectories associated with policy π and context θ.

At test time, we are interested in behaving optimally in a CMDP $\mathcal{M}(\theta_{\text{test}})$ allocated according to the prior $\theta_{\text{test}} \sim p(\theta)$. Like other successor feature-based approaches [Barreto et al., 2017, Brown and Niekum, 2019, Filos et al., 2021, Janz et al., 2019] we make the following assumption about the reward parameterization:

Assumption 2. *The underlying reward function is bounded and can be represented as a linear function with respect to a reward feature vector $\nu(s, a)$ that is known a priori such that $r(s, a; \omega^{\star}) = \nu(s, a)^{\top}\omega^{\star} \in [r_{min}, r_{max}]$. Furthermore, the reward function is shared across all CMDPs, i.e., independent of θ.*

This assumption simplifies our analysis and enables the derivation of convex optimization procedures. We do not assume oracle access to perfect reward features. Instead, the features could be the result of an unsupervised learning step external to our algorithm. Crucially, the agent can observe states and choose actions according to a policy but does *not* observe rewards nor θ_{test} and does not know the true reward parameterization $\omega^\star$. Instead, the agent is given a dataset of N_{Expert} demonstrations $\mathcal{D}_{\text{Expert}} \coloneqq \{\tau_i\}_{i=1}^{N_{\text{Expert}}}$ of state-action trajectories $\tau_i \coloneqq \{s_0, a_0, s_1, a_1, \dots\}$ of length H_i in CMDPs sampled from $p(\theta)$, where each expert $\pi^\star_{\text{Expert}}(\cdot, \theta_i)$ behaves optimally in its assigned CMDP, e.g., expert trajectories of the robot navigating different mazes where the ledge locations differ. Furthermore, the agent has access to a simulator, where CMDPs are allocated according to the prior $p(\theta)$, and the agent can interact with the corresponding environment, observing a trajectory of state-action pairs. We denote the complete dataset of simulated trajectories as $\mathcal{D}_{\text{Simulator}} \coloneqq \{\tau_i\}_{i=1}^{N_{\text{Simulator}}}$. The agent never directly observes rewards nor θ_i in either $\mathcal{D}_{\text{Expert}}$ or $\mathcal{D}_{\text{Simulator}}$.

At test time, the agent is assigned a CMDP according to $\theta_{\text{test}} \sim p(\theta)$ and interacts with $\mathcal{M}(\theta_{\text{test}})$, obtaining a history of state-actions: $h_t \coloneqq \{s_0, a_0, \cdots a_{t-1}, s_t\}$ at time t and takes actions according to a history-conditioned policy: $a_t \sim \pi(h_t)$. The agent never observes the reward history.

8.2.1 The Tiger-Treasure Problem

We introduce a variant of the classic "Tiger-Treasure problem" from Kaelbling et al. [1998] to illustrate how naïvely applying imitation learning fails when there is an imitation gap. Consider the CMDP in figure 8.1 indexed by $\theta \in \{1, 2\}$, representing which door the tiger is behind. The prior is $p(\theta = 1) = p(\theta = 2) = 0.5$. In both CMDPs, the agent starts in state S_0 and the set of actions available is $\mathcal{A} = \{o_1, o_2, \text{listen}\}$, where o_1 and o_2 open the corresponding door and 'listen' listens for a tiger. In any state $s \in \{S_0, T_1, T_2\}$, the agent transitions deterministically to state Tiger if $a = o_1$ and $\theta = 1$ or $a = o_2$ and $\theta = 2$, and conversely for the Gold state. The goal of the agent is to reach the treasure (labeled 'Gold') whilst avoiding the Tiger. The agent receives a reward $r(\text{Gold}, \cdot) = 10$ for finding the

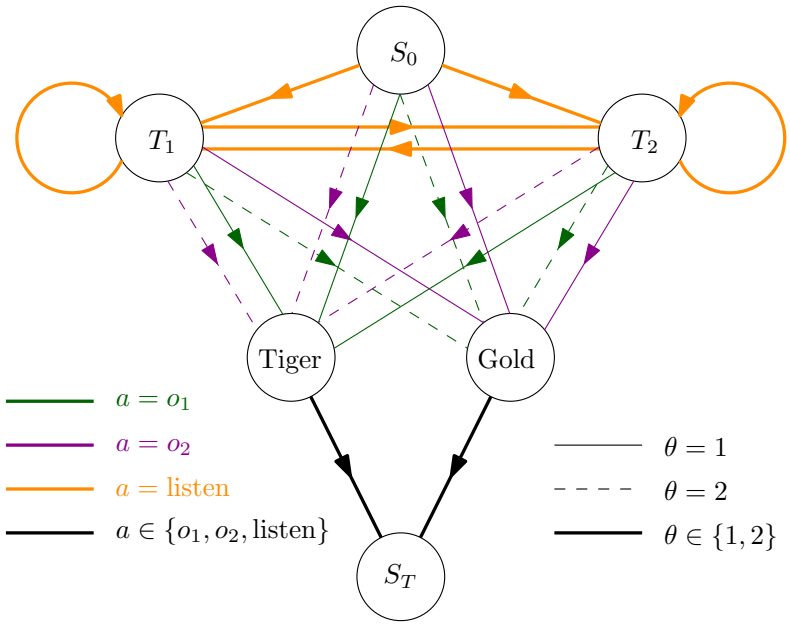


Figure 8.1: A diagram of the Tiger-Treasure problem MDP, a classic example of an imitation gap. The agent initially does not know which door the treasure or tiger is behind and must take listening actions to resolve its uncertainty.

gold and $r(\text{Tiger}, \cdot) = -100$ for finding the tiger. After this, the agent transitions to terminal state S_T regardless of action taken.

The agent can also listen before any doors are opened, receiving a stochastic signal with success rate $p > 0.5$ correlated with the identity of the door with the tiger. Hearing the tiger in room i transitions the agent to state T_i. For $s \in \{S_0, T_1, T_2\}$ and $a = \text{listen}$, the agent transitions to state T_1 with probability p if $\theta = 1$ and T_2 with p if $\theta = 2$, otherwise it transitions to state T_2 with probability $1 - p$ if $\theta = 1$ and T_1 with $1 - p$ if $\theta = 2$. States T_1 and T_2 are not present in Kaelbling et al. [1998]'s original problem, but are included here because reward is assumed independent of θ and hence partial observability about the MDP is encoded in the state. Entering a listening state incurs a small negative reward $r(T_1, \cdot) = r(T_2, \cdot) = -1$. All other rewards not specified are 0.

For ease of exposition, assume the listening success $p = 1$. The expert has privileged knowledge about the MDP, and always deterministically chooses to open the door with the gold behind it. This eliminates the need for the expert to explore resulting in expert trajectories where the listening states are never visited. At test

time, there is an imitation gap, as the agent does not know a priori which door the tiger is behind. A naïve imitator does not realize that the expert is conditioning on extra information and so thinks the expert is randomly choosing which door to open; imitating that gives suboptimal return of -45γ. By contrast, an agent that chooses to listen always receives a return of $10\gamma - 1$ if it acts optimally on the revealed location of the tiger. This example demonstrates the failure of naïve imitation learning in simple settings when there is an imitation gap.

8.3 Towards a Bayesian Solution

In this section, we propose a **B**ayesian solution to the **I**mitation **G**ap (BIG), where the goal is to learn a policy that can optimally trade-off its uncertainty at test time with imitating the expert demonstrator. Unlike in the canonical Bayesian RL setting, our formulation does not have access to reward samples from which to infer the underlying reward function. Instead, the reward prior determines the cost of exploration (COE) for the agent at test time, meaning that the agent can still behave optimally under partial observability. We provide an overall schematic of our approach in figure 8.2.

BIG has three main phases which are labeled in figure 8.2. In the first, we provide an initial prior over the reward parameterization $p(\omega)$. We integrate information from the expert data $\mathcal{D}_{\text{Exp}}$ into the prior using a novel *contextual* Bayesian IRL (BIRL) approach (sections 8.3.1 and 8.3.2) that infers the posterior $p(\omega|\mathcal{D}_{\text{Exp}})$. Because entire classes of reward functions can explain expert data equally well, IRL is an underspecified problem and approaches can equally penalize any unvisited state.

In the second phase (section 8.3.3), we restrict the class of reward functions to those that allow for exploration at test time, by first normalizing the posterior so that the predictive reward lies in the range $[r_{\min}, r_{\max}]$ according to assumption 2. Representing reward as $r = k \times r_{\max}$, we specify a cost of exploration prior $p(k)$ over rewards at key state-action pairs unseen in the expert demonstrations to integrate the cost of exploration information at test time. This approach ensures that all rewards still belong to the same *class* [Rafailov et al., 2023, Definition 1] after the

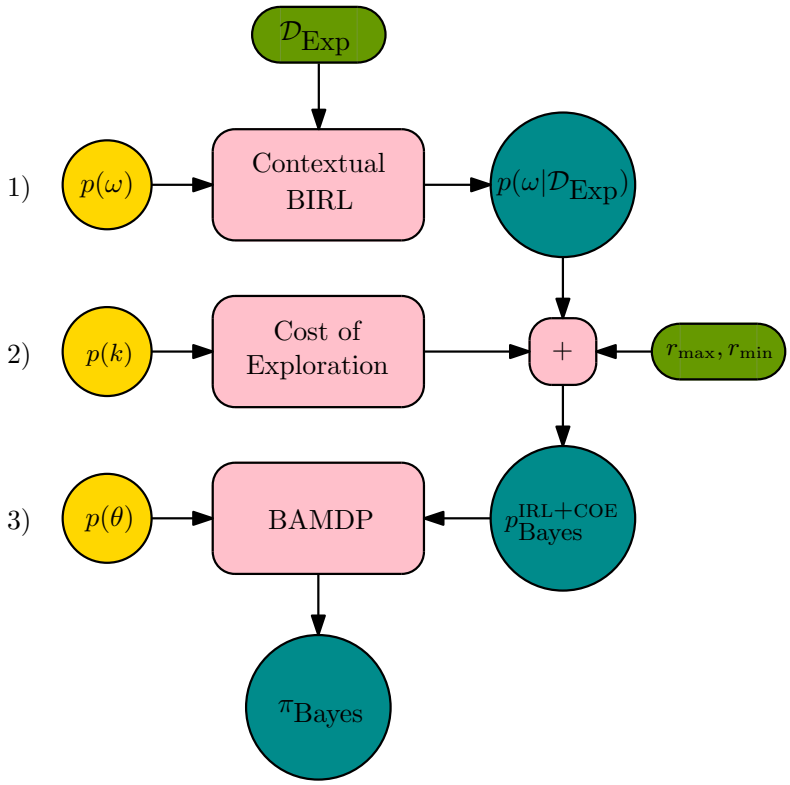


Figure 8.2: Schematic of the Bayesian solution to the Imitation Gap (BIG). Prior information is shown in **green**, algorithms in pink, prior distributions in **yellow**, and outputs in **teal**.

IRL stage, i.e., yielding an equivalent optimal policy. This step is required as the expert demonstrations do not have full coverage. $p(k)$ encodes the relative cost of deviating from an optimal policy which encourages exploration for the downstream policy, complementing the IRL data. Integrating both the IRL and COE priors, we denote the Bayesian reward distribution as $p^{\text{IRL+COE}}_{\text{Bayes}}(r|s,a)$. We present pseudocode for an algorithm implementing the first and second phase in appendix D.4.

In the third phase, we solve a Bayesian RL problem in which the goal is to learn a Bayes-optimal policy π_{Bayes} that can be deployed at test time. As shown in figure 8.2, the inputs to the Bayes-adaptive MDP (BAMDP) are the Bayesian reward distribution $p^{\text{IRL+COE}}_{\text{Bayes}}(r|s,a)$ and a prior over context variables $p(\theta)$ (section 8.3.4). The agent extracts reward information from the expert trajectories to learn a predictive reward but does not directly imitate the expert's behavior and thus can adapt to the unknown MDP at test time, avoiding the problems with naïve imitation learning discussed in section 8.2.1. Due to the diverse nature of the

sources of input information, inferring the reward posterior requires the agent to learn and maintain several distributions over random variables, we summarize them in table D.1 in appendix D.1.

8.3.1 Contextual Successor Features

Before performing BIRL, we must learn a contextual value function representation to define the likelihood over expert trajectories. For an agent following policy π, we can characterize the expected discounted return as a function of state-action pairs via the contextual Q-function $Q^\pi(s, a, \theta, \omega)$, which satisfies the contextual Bellman equation: $\mathcal{B}^\pi[Q^\pi](s, a, \theta, \omega) = Q^\pi(s, a, \theta, \omega)$ where $\mathcal{B}^\pi[Q^\pi](s, a, \theta, \omega)$ is the contextual Bellman operator: $\mathcal{B}^\pi[Q^\pi](s, a, \theta, \omega) \coloneqq \nu(s, a)^\top \omega + \gamma \mathbb{E}_{s' \sim p(s'|s,a,\theta) a' \sim \pi(a'|s')} \left[Q^\pi(s', a', \theta, \omega) \right]$.

Assumption 2 specifies a linear reward function, which allows us to use a *successor feature* representation of the Q-function that factors the reward parameterization out of $Q^\pi(s', a', \theta, \omega) = \Psi^\pi(s', a', \theta)^\top \omega$ where $\Psi^\pi(s, a, \theta)$ is the contextual successor feature, defined as: $\Psi^\pi(s, a, \theta) = \mathbb{E}_{\tau \sim p^\pi(\tau_\infty|\theta)} \left[\sum_{t=0}^{\infty} \gamma^t \nu(s_t, a_t) \middle| s_0 = s, a_0 = a \right]$. N.b., the reward is linear w.r.t. the features $\nu(s, a)$, which could themselves be learned and arbitrarily complex. Learning $\Psi^\pi(s, a, \theta)$ means that we do not need to solve a Bellman equation every time ω changes; we can simply take a dot product between the existing successor feature and the new ω. Appendix D.2 details the training process.

8.3.2 Contextual Bayesian IRL

We now describe the first phase of our pipeline in figure 8.2. Inferring the reward function is a Bayesian regression problem. As is typical for regression problems [Murphy, 2013], we specify a Gaussian reward model $p(r|s, a, \omega) = \mathcal{N}(\nu(s, a)^\top \omega, I\sigma^2)$ with mean $\nu(s, a)$ and scalar variance parameter $\sigma \in \mathbb{R}$. We specify a Gaussian prior over the unknown reward parameterization $\mathcal{N}(\omega|\omega_0, I\sigma_0^2)$ where ω_0 is the prior mean and prior variance $\sigma_0^2 > 0$ represents the belief in ω_0. Tuning σ_0^2 thus allows us to set how much the expert's trajectories affect the prior reward specification after BIRL. We now exploit expert data to refine our prior by leveraging approaches from BIRL.

To derive the likelihood, it is necessary to define a model of the expert observations. In the classic Bayesian IRL approach [Ramachandran and Amir, 2007], a likelihood is specified for a single MDP: $p(\tau_i|\omega)$. To adapt our approach to experts that act in multiple MDPs, our likelihood should account for the context $p(\tau_i|\theta_i, \omega) = \prod_{j=0}^{T_i-1} p(s_j, a_j|\theta_i, \omega)$. We assume that the agent's policy is represented by a potential function defined by the optimal Q-function for the agent's MDP:

$$p(a|s, \theta_i, \omega) = \frac{1}{z(s, \omega, \theta_i)} \exp\left(\frac{1}{\alpha}\Psi(s, a, \theta_i)^\top \omega\right), \tag{8.1}$$

where $z(s, \omega, \theta)$ is the normalization constant. This can be viewed as a Boltzmann distribution (or a softmax-like function) over the actions. Here α is a temperature parameter that controls how optimal the expert's actions are with respect to $Q^\star(s, a, \theta_i, \omega) = \Psi^\star(s, a, \theta_i)^\top \omega$. As is convention [Arora and Doshi, 2021, Ramachandran and Amir, 2007], this assumption stabilizes inference algorithms by softening the Dirac delta policy that the agent is following, allowing for gradients to flow into the density. A deterministic optimal policy is recovered in the limit $\alpha \to 0$.

Given the likelihood and prior, we infer the expert reward posterior $p(\omega|\mathcal{D}_{\text{Expert}})$. Marginalizing, we obtain the Bayesian reward distribution:

$$p^{\text{IRL}}_{\text{Bayes}}(r|s, a) \coloneqq \mathbb{E}_{\omega\sim p(\omega|\mathcal{D}_{\text{Expert}})}\left[p(r|s, a, \omega)\right], \tag{8.2}$$

which incorporates both the epistemic uncertainty from the posterior and the aleatoric uncertainty from the reward model. Analogously to Bayesian logistic regression [Murphy, 2013], using the potential function from equation (8.1) does not yield a closed-form solution for the posterior. Instead, we use the Laplace approximation for the posterior, which yields a stochastic gradient descent update:

Theorem 2. *Define* $\varsigma_0^2 \coloneqq \frac{\sigma_0^2}{\alpha}$. *Using the Laplace approximation, the approximate posterior is* $p(\omega|\mathcal{D}_{Expert}) \approx \mathcal{N}(\omega|\omega^\star_{Laplace}, \Sigma_{Laplace})$ *where* $\Sigma_{Laplace} = \nabla^2_\omega \log p(\omega^\star_{Laplace}|\mathcal{D}_{Expert})$ *and* $\omega^\star_{Laplace}$ *is the MAP estimate, which can be found by carrying out the following stochastic gradient descent updates on the log-posterior:*

$$\omega \leftarrow \omega + \eta_\omega\left(N_{Expert} H_i\left(\Psi^\star(s_j, a_j, \theta_i) - \mathbb{E}_{a_i\sim p(a_i|s_j,\omega,\theta_i)}\left[\Psi^\star(s_j, a_i, \theta_i)\right]\right) - \frac{(\omega - \omega_0)}{\varsigma_0^2}\right). \tag{8.3}$$

Proof. See appendix D.6. □

The Bernstein von-Mises theorem formally justifies the approximation [Doob, 1949, van der Vaart, 1998], proving that under mild regularity assumptions, the posterior tends to the Laplace approximation in the limit of increasing data. ς_0^2 controls how the prior influences learning; as $\varsigma_0^2 \to 0$, expert data is ignored and $\omega^\star_{\text{Laplace}} = \omega_0$. The posteriors over each expert's contextual variable $p(\theta_i|\tau_i)$ typically have no closed-form analytic solution. We can approximate each $p(\theta_i|\tau_i)$ using variational inference instead, as detailed in appendix D.5.

Role of Temperature. As IRL is underspecified, many reward functions can explain the expert data. A key insight from theorem 2 is the role of the temperature parameter α in determining the relative difference between the lowest and highest rewards assigned. The example in appendix D.5 illustrates that when $\alpha \to 0$, the expert policy model becomes more deterministic, always choosing the action with the highest return. Arbitrarily small differences between rewards explain expert behavior, so IRL pulls the parameterization difference to be as small as possible. Conversely, when $\alpha \to \infty$, the expert policy becomes more stochastic, taking low-return actions more frequently in proportion to their value. An increasingly large separation in rewards is needed to explain expert behavior.

8.3.3 Cost of Exploration Prior

For the second phase, we specify a prior over the reward to incorporate information on the cost of exploration into the posterior reward returned by the contextual BIRL to refine the solution. Note that the imitation gap problem would not be solvable without access to prior information defining the cost of exploration, as the expert data does not demonstrate which states are safe to explore and how to balance exploration and exploitation. This prior is only introduced for exploration and does not need to contain any information about the exploitation, as that is learned from the expert demonstrations. Furthermore, the prior could be arbitrarily uninformative, in the third phase, our method learns the Bayes-optimal policy for any prior.

As shown in figure 8.2, we start by rescaling the posterior reward to be within the bounds $[r_{\min}, r_{\max}]$ according to Assumption 2. We denote the corresponding scaled reward parameterization as $\bar{\omega}^{\star}_{\text{Bayes}}$. Given the infinite horizon, any linear transformation applied to the reward function results in the same optimal policy.

We assume that we know a subset of state-action pairs denoted as $\mathcal{C} = \{(s,a)|s \in \mathcal{S}_{\text{COE}}, a \in \mathcal{A}_{\text{COE}}(s)\}$ where exploration can be performed. We illustrate this for the Tiger-Treasure problem from figure 8.1; as rewards only depend on the state in this problem, the set is $\mathcal{S}_{\text{COE}} = \{T_1, T_2\}$. We introduce a simple COE inside the rescaled IRL reward parameterized by $\bar{\omega}^{\star}_{\text{Bayes}}$ by specifying a reward function $p(r|s,a,k) = \mathcal{N}(r|kr_{\max}, \sigma^2)$ over $(s,a) \in \mathcal{C}$ for some $k \in [\frac{r_{\min}}{r_{\max}}, 1)$; this ensures the mean is contained in $[r_{\min}, r_{\max}]$. A scale k that varies across action-state pairs may also be specified if a more complex cost of exploration information needs to be modeled. In the simple Tiger-Treasure problem, we know from construction the set of states where exploration can be performed. In a practical setting with large state space, these states could be obtained by considering non-expert, but safe behaviors from other policies acting in the same environment. This could require density estimation for continuous state spaces. We leave development of task specific cost of exploration priors for future work and focus on demonstrating the general principles in this work.

The value of k determines how risk-averse the agent is at test time. For $k \approx 1$ the agent values exploratory state-actions in $\mathcal{C}$ nearly as much as the most rewarding state-actions learned from IRL. As such, the cost of exploration is low, and the agent explores until it is highly certain about avoiding low reward actions in the imitation gap, encouraging conservative behavior. Conversely, as $k \to \frac{r_{\min}}{r_{\max}}$ the agent becomes less risk-averse and recovers a purely behavioral cloning regime, taking actions that could lead to low reward as they have similar value to exploratory actions.

To incorporate epistemic uncertainty in k, we specify a prior $p(k)$ with support over $[\frac{r_{\min}}{r_{\max}}, 1)$. Marginalizing, we obtain the Bayesian reward distribution over $\mathcal{C}$:

$$p^{\text{COE}}_{\text{Bayes}}(r|s,a) = \mathbb{E}_{k\sim p(k)}\left[p(r|s,a,k)\right]. \tag{8.4}$$

For all other state-action pairs, the Bayesian reward distribution remains unchanged, yielding the distribution over $\mathcal{C}$:

$$p^{\text{IRL + COE}}_{\text{Bayes}}(r|s,a) = \begin{cases} p^{\text{COE}}_{\text{Bayes}}(r|s,a) & s,a \in \mathcal{C}, \\ p^{\text{IRL}}_{\text{Bayes}}(r|s,a) & \text{otherwise.} \end{cases} \tag{8.5}$$

8.3.4 Bayes-Optimal Policy Learning

For the third phase, we perform Bayesian reinforcement learning, which optimally trades off exploration and exploitation by conditioning actions on the agent's uncertainty over θ. We can define a Bayes-adaptive MDP [Duff, 2002, BAMDP] using the contextual MDP in equation (2.2) as a model. At test time, the agent is assigned an MDP $\theta_{\text{test}} \sim p(\theta_{\text{test}})$ and can observe samples from $p(s'|s,a,\theta_{\text{test}})$ by interacting with the MDP via its policy. A history of interactions is denoted $h_t \coloneqq \{s_0, a_0, s_1, a_1, \dots s_t\} \in \mathcal{H}_t$ where $\mathcal{H}_t$ is the corresponding state-action product space. After observing a history h_t from the assigned MDP, the agent updates its belief in θ_{test} according to the posterior:

$$p(\theta_{\text{test}}|h_t) = \frac{\prod_{i=1}^{t} p(s_t|s_{t-1},a_{t-1},\theta_{\text{test}})p(\theta_{\text{test}})}{\mathbb{E}_{\theta_{\text{test}}\sim p(\theta_{\text{test}})}\left[\prod_{i=1}^{t} p(s_t|s_{t-1},a_{t-1},\theta_{\text{test}})\right]}. \tag{8.6}$$

Using the posterior, we can define the Bayesian transition distribution as:

$$p(s_{t+1}|h_t,a_t) = \int_{\Theta} p(s_{t+1}|s_t,a_t,\theta_{\text{test}})p(\theta_{\text{test}}|h_t)d\theta_{\text{test}}. \tag{8.7}$$

As there is no reward signal available to the agent and rewards do not depend on θ, the Bayesian reward distribution in the BAMDP is exactly the combined Bayesian reward distribution from equation (8.5): $p(r_t|h_t,a_t) = p^{\text{IRL + COE}}_{\text{Bayes}}(r_t|s_t,a_t)$. We denote the joint reward-state Bayesian transition distribution as $p(r_t, s_{t+1}|s_t,a_t) = p(r_t|h_t,a_t)p(s_{t+1}|h_t,a_t)$, which is equivalent to the predictive trajectory transition distribution: $p(\tau_{t+1}|\tau_t,a_t) = p(\tau_t,a_t,r_t,s_{t+1}|\tau_t,a_t) = p(r_t,s_{t+1}|h_t,a_t)\underbrace{p(\tau_t,a_t|\tau_t,a_t)}_{=1} = p(r_t,s_{t+1}|h_t,a_t)$. Here $p(\tau_{t+1}|\tau_t,a_t)$ is used to reason over unobserved counterfactual trajectories, and so must account for predictive reward. We define the corresponding BAMDP similarly to Fellows et al. [2023a] as $\mathcal{M}_{\text{BAMDP}} \coloneqq \langle \mathcal{T}, \mathcal{A}, p(\tau_{t+1}|\tau_t,a_t), p(s_0), \gamma \rangle$ where $\mathcal{T}$ is the space of all possible trajectories.

In Bayesian RL, policies $\pi_{\text{Bayes}}(a_t|h_t)$ map from *histories* to distributions over actions, and our goal is to learn a Bayes-optimal policy $\pi^{\star}_{\text{Bayes}}(a_t|h_t)$ that solves $\mathcal{M}_{\text{BAMDP}}$. Due to the linearity of our formulation, we show in appendix D.4.1 that solving the BAMDP is equivalent to optimizing the following Bayesian RL objective for π^{Bayes}:

$$J^{\pi}_{\text{Bayes}} = \mathbb{E}_{p(\theta_{\text{test}})}\left[\mathbb{E}_{p(h_{\infty}|\theta_{\text{test}})}\left[\sum_{i=0}^{\infty}\gamma^{i} r^{\text{IRL + COE}}_{\text{Bayes}}(s_i, a_i)\right]\right], \tag{8.8}$$

where $p(h_{\infty}|\theta_{\text{test}}) = p_0(s_0)\prod_{i=0}^{\infty} p(s_{i+1}|s_i, a_i, \theta_{\text{test}})\pi(a_i|h_i)$, and $r^{\text{IRL + COE}}_{\text{Bayes}}(s, a)$ is the predictive reward:

$$r^{\text{IRL + COE}}_{\text{Bayes}}(s, a) \coloneqq \begin{cases} r_{\max}\mathbb{E}_{k\sim p(k)}\left[k\right] & s, a \in \mathcal{C}, \\ \nu^{\top}(s, a)\bar{\omega}^{\star}_{\text{Bayes}} & \text{otherwise.} \end{cases} \tag{8.9}$$

In the following empirical evaluation, we approximate Bayes-optimal policies by training a policy conditioned on the true inference model using DQN [Mnih et al., 2013].

8.4 Empirical Evaluation

To evaluate BIG, we conduct experiments across a series of imitation gap problems. In all the tested environments, the agent has to solve a task that requires exploration. First, we demonstrate that in the Tiger-Treasure environment, naïve IRL learns a reward function that does not lead to the desired exploratory policy, whereas BIG does by exploiting prior information about the cost of exploration. Second, we illustrate that, by marginalizing over the context distribution, naïve IRL can learn a reward function that does not capture the expert's intent. Finally, we present results in a gridworld environment to show that BIG can handle imitation learning tasks with larger state-action spaces. Since we assume no access to true environment reward or expert state information, most previous solutions to the imitation gap are not applicable. Instead, we compare to maximum entropy IRL (labeled 'No-Prior' in our experiments). For convenience, we implement algorithm 6 using the true posterior $p(\theta|\tau)$ as the inference model. In all experiments, we use deep neural networks for Ψ. We assume normally distributed errors and report standard error across seeds in the figures.

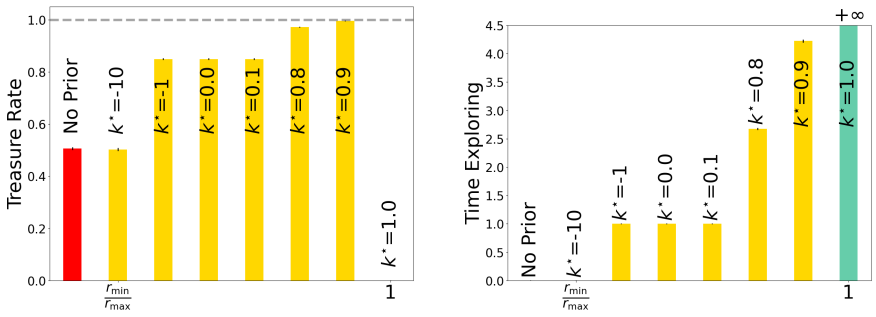


Figure 8.3: Evaluation of BIG in the Tiger-Treasure environment. Success rate and time exploring (in steps) for policies learned with a uniform prior reward with different means are represented in yellow ($k^\star < 1$) and in green ($k^\star = 1$), while the case with no prior is shown in red. Error bars indicate the standard error of the mean across 10 seeds. The symbol $+\infty$ indicates that, for some trials, the agent explores throughout the entire (infinite) episode.

8.4.1 Investigating Reward Priors

In the Tiger-Treasure environment introduced in section 8.2.1, since the expert always goes to the treasure room directly, we cannot extract information about optimal exploration from the expert data. As discussed in section 8.3.3, we use the prior $p(k)$ to enable exploratory behavior at test time. We explore the influence of this reward prior on the environment from figure 8.1, using a space of uniform priors $p(k) = \text{Unif}([a, b])$ over intervals $[a, b] \in \left[\frac{r_{\min}}{r_{\max}}, 1\right)$. We choose $r_{\min} = -100$ and $r_{\max} = 10$. While these bounds are arbitrary, they reflect the undesirability of failing in the task demonstrated by the expert. The reward feature ν is a one-hot indicator over the states. Figure 8.3 presents the agent's success rate in reaching the treasure, along with the average time exploring, which corresponds to the number of listening actions, for different values of the prior mean $k^\star$. As expected from section 8.3.3, when $k^\star$ approaches 1, the agent explores more, never exploiting when $k^\star = 1$. The probability of reaching the treasure increases with the number of listening actions. Therefore, as $k^\star$ approaches 1 (without reaching it), the treasure rate also increases. By contrast, when $k^\star$ decreases the agent begins to listen less often. These behaviors correspond to the Bayes-optimal policies for each of the priors, demonstrating that our method learns the Bayes-optimal policy irrespective of the prior supplied.

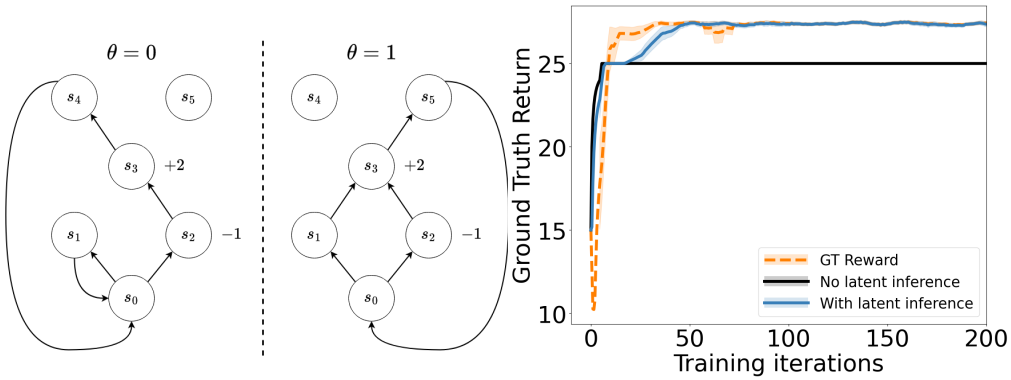


Figure 8.4: A demonstration of the necessity for latent inference with BIG. On the left, we show the CMDP used in the experiments, with two possibilities for the context θ. On the right, we show the ground truth returns of a DQN agent for trajectories of 100 steps in the CMDP during training. The shading shows the standard error of the mean for 8 seeds.

8.4.2 Investigating Latent Inference

Next, we look at whether inferring the latent θ during IRL matters for learning the desired reward function. We experiment in a simple CMDP environment depicted in figure 8.4. For a mathematical example, see appendix D.3. In this environment, the expert policy goes to the state s_3, which provides a reward of $+2$ and then loops back to state s_0 through s_4 or s_5. It prefers to take the route through s_1, when it is available, to avoid the negative reward of -1 in s_2. When θ is distributed such that $p(\theta = 0) > p(\theta = 1)$, it can lead to naïve IRL misidentifying the expert intent. To see why, consider that to fit the expert behavior without information about θ, the reward function has to make the path through s_2 more likely in both MDPs. Conditioning the successor features on the inferred θ resolves this issue, because then identifying the reward reduces to standard IRL in two separate MDPs. To verify this in practice, we show results of learning the reward functions with and without latent inference in figure 8.4. We choose the latent to be distributed as $p(\theta = 0) = 0.9$. No reward prior is used. The policy trained on the rewards learned with latent inference matches the policy trained with the ground truth rewards. The rewards learned without latent inference do not result in as good a policy, demonstrating that latent inference is necessary to learn the correct rewards in a general CMDP. See appendix D.6.2 for an analysis of the learned rewards.

8.4.3 Reward Priors in a Larger CMDP

Finally, we test BIG in a grid-world environment with pixel-based observations. The agent observes a top-down view of the environment similar to the illustration of the learned rewards presented in figure 8.5. The agent can move in four directions and take listening actions. Taking the listening action in any grid cell results in a stochastic transition to a state, which indicates the location of the gold, but otherwise has the same dynamics as the cell that the action was taken in. When the agent enters a door, it is moved to Tiger or Gold depending on θ. From those states, the agent is moved to the grid cell marked with x in the next timestep, setting the agent up for another round in the maze. The expert takes the shortest path from any state to Gold, choosing the correct door, depending on θ. See appendix D.6.4 for details.

Three DQN training curves are shown in figure 8.5 corresponding to a manually constructed ground truth reward, which explains the expert behavior, reward learned by the contextual IRL without using a reward prior, and the same reward refined using the prior. These learning curves show that BIG with a particular reward prior produces a similar BAMDP policy as the manually constructed reward. At the same time, using just the IRL reward results in a policy that initially chooses a door at random. We compare multiple values for $k^\star$ and find that as in the simple Tiger-Treasure environment, different values result in over-exploration or under-exploration leaving a range in the middle that results in similar exploration as the handcrafted reward. This experiment shows that BIG can recover a reward function, which enables the agent to complete the task demonstrated by the expert despite a challenging imitation gap.

8.5 Related Work

We provide a broader survey of related work in IL and imitation gap in chapter 6. Here, we focus on related work that is specific to the work presented in this chapter.

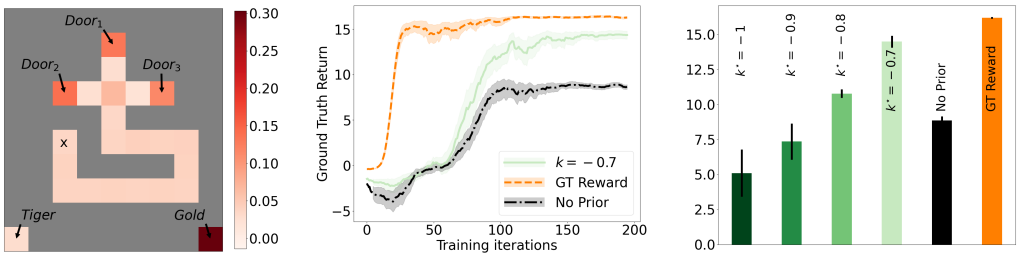


Figure 8.5: BIG successfully learns the optimal behavior in a challenging gridworld environment. On the left, we show the rewards learned by the contextual IRL. In the middle, we show the return (using the manually constructed reward) of policies trained with reward inferred with and without a reward prior and the manually constructed reward (ground truth). The shading shows the standard error of the mean for 8 random seeds. On the right, we show the final returns of policies trained using different values of $k^\star$ compared to not using the reward prior and using the ground truth reward.

Bayesian RL Our work shares similar components to other Bayesian approaches to RL. For instance, VariBAD and related methods [Zintgraf et al., 2020, 2021] consider a model-based approach for learning approximations to Bayes-optimal policies by exploiting meta-reinforcement learning [Beck et al., 2023] to perform inference over a subset of the unobserved context. Similarly, BEN [Fellows et al., 2023b] is a model-free approach that learns Bayes-optimal policies by specifying a model and prior over the optimal Bellman operator. All of these methods assume access to the reward and do not consider the issue of imitating an expert. Bayesian Inverse RL approaches infer a posterior over the unknown reward distribution given trajectories of demonstrations [Ramachandran and Amir, 2007] (see Adams et al. [2022], Table 1 for a complete list of existing approaches). BIG generalizes these approaches in section 8.3.2 to account for the unobserved context variable before integrating the learned reward posterior into a BAMDP.

Successor Features Successor features [Barreto et al., 2017, Brown and Niekum, 2019] provide an elegant representation of value functions under the assumptions of a linear reward function. Janz et al. [2019] incorporate successor features into the Bayesian framework. Most similar to BIG, successor features have been used successfully in Psi-Phi Learning [Filos et al., 2021] for multi-task

inverse reinforcement learning. Psi-Phi Learning can also retrieve the reward parameterizations for a new agent (expert) but does so with full observability.

8.6 Limitations

To make the empirical setup close to the theory, we use ground truth inference models and linearity assumptions, which means that scaling up the algorithm requires revisiting those choices. For example, by learning an approximate inference model. In the CMDPs considered in this chapter, we assumed the set of allowed exploration states was explicitly known a priori, which may be a limiting assumption in harder problems. As is typical in Bayesian methods, we leave the design of problem-specific priors to the practitioners focused on those problems.

8.7 Conclusion

In this chapter, we proposed a fully Bayesian solution to the imitation gap which integrated various priors over the reward parameterization, cost of exploration, and hidden state. This allowed us to derive a BAMDP formulation of the problem whose solution optimally trades off exploration and exploitation at test time. We demonstrated the importance of each component of our algorithm across a series of experiments with an imitation gap before showing that BIG scales to larger maze problems, including those with high-dimensional pixel-based observations. Crucially, in contrast to previous work, no online reward information was required. This makes our work particularly exciting for challenging imitation problems where no reward is easily specifiable, and extending BIG to even more complex settings is a promising direction for future work. In the next chapter, we propose one such more complex imitation gap setting in the form of a benchmark based on data from autonomous driving.

8.8 Statement of Authorship

Title of Paper: A Bayesian Solution To The Imitation Gap

Publication Status

- Published
- Accepted for Publication
- Submitted for Publication
- Unpublished and unsubmitted work written in a manuscript style ✓

Publication Details: Vuorio, Risto, Mattie Fellows, Cong Lu, Clémence Grislain, and Shimon Whiteson. "A Bayesian Solution To The Imitation Gap." *arXiv preprint arXiv:2407.00495* (2024).

Contribution to the paper: I was the co-lead author for the paper. I helped steer the overall research direction, implemented algorithms, conducted experiments, and helped write the manuscript.

9 Conclusion

In this thesis, we have explored significant advancements in the fields of meta-RL and IL, two domains at the forefront of developing more interactive artificial intelligence. Through a series of original contributions, we addressed challenges in learning agents for dynamic scenarios rather than static generation problems, paving the way toward more adaptive and intelligent agents.

The first part of this thesis focused on meta-RL, specifically on improving the sample efficiency of RL algorithms. We demonstrated how meta-gradients could be effectively used to adapt credit assignment strategies, facilitating improved learning efficiency in complex environments with delayed rewards. Additionally, we presented a comprehensive survey of meta-RL methods and provided theoretical and empirical insights into the challenges of estimating meta-gradients, particularly the tradeoff between bias and variance.

The second part of the thesis tackled the imitation gap problem within IL, where a disparity exists between the observations of the expert and the imitator. We proposed new algorithms for learning in settings where the expert has access to information that the imitator does not, modeling this discrepancy as confounding latent variables or leveraging Bayesian priors to guide policy learning. These methods were shown to outperform existing approaches in imitation gap scenarios, effectively mitigating issues caused by the missing information.

The work presented in this thesis has not only advanced the state-of-the-art in meta-RL and IL but has also opened several promising directions for future research. Below, we outline some potential areas that could benefit from further exploration:

Improving Meta-Gradient Estimators While meta-gradient methods have shown promise in dynamically adapting learning strategies, challenges remain in accurately estimating these gradients, especially in high-dimensional environments and over longer horizons. Developing more efficient meta-gradient estimators that strike a better balance between computational feasibility and gradient accuracy is a key future goal.

Adaptive Learning in Real-World Applications While this thesis focused on simulated environments, the extension of meta-RL algorithms to real-world settings, such as robotic control and autonomous driving, poses exciting challenges. Real-world scenarios involve high-dimensional observations, complex reward structures, and safety constraints that meta-RL algorithms must handle. Investigating how these methods can be effectively deployed, will be critical for bridging the gap between simulation and real-world application.

Developing More Scalable Algorithms for the Imitation Gap The contributions in thesis demonstrate the feasibility of dealing with causal confounders and the lack of exploration information through variational inference and Bayesian IRL. However, the empirical results presented in this thesis are limited to relatively small problems. Future research could explore scaling up the approaches presented in this thesis, possibly by substituting the theoretically principled but expensive to run algorithms with more scalable alternatives employing futher abstractions.

Unifying Advances in meta-RL and IL This thesis presented contributions in the two closely related but disjoint fields of meta-RL and IL. Previous work in meta-IL exists, but has not been adapted to the imitation gap setting. While the motivations for meta-IL and IL algorithms for the imitation gap differ, both problems

require learning imitators for distributions of tasks. More research exists on meta-IL than the imitation gap. Therefore, a highly promising avenue of future work is to take approaches from the meta-IL literature and adapt them to the imitation gap setting.

In summary, this thesis has presented novel contributions to both meta-reinforcement learning and imitation learning, offering new algorithms, empirical analyses, and theoretical insights. The work lays a foundation for more interactive and adaptive AI systems that learn effectively from their environment and from experts. By addressing the challenges of sample efficiency in RL and the imitation gap in IL, this thesis contributes to the broader vision of creating intelligent agents capable of robust decision-making and learning in complex, real-world scenarios. The future directions outlined above highlight the exciting potential for continued research in this evolving field.

Appendices

A

Adaptive Pairwise Weights for Temporal Credit Assignment

Contents

A.1 Metagradient Algorithm

In this section we give further details on the metagradient algorithm used to learn the pairwise weights for both Meta-PWTD and Meta-PWR as well as the metagradient

Algorithm 2 A generic metagradient algorithm.

Require: Initial parameters η_0 and ϕ_0.
  Sample a batch of trajectories τ_0 with η_0;
  repeat
    Set $\eta_{k+1} = \eta_k + \alpha^\eta \nabla_\eta J^{\text{inner}}(\eta_k, \tau_k; \phi_k)$
    Sample a batch of trajectories τ_{k+1} with η_{k+1}
    Set $\phi_{k+1} = \phi_k + \alpha^\phi \nabla_\phi J^{\text{outer}}(\eta_{k+1}, \tau_{k+1})$
    $k \leftarrow k + 1$
  until done

baselines. We use the algorithm from Xu et al. [2018]. A generic metagradient algorithm is presented in Algorithm 2 where we have extended the syntax from the chapter to include the trajectories and meta-parameters in the inputs of the objective functions to make the dependencies more explicit. For clarity, the algorithm omits the value function update steps, which differ between the specific algorithms as explained in the chapter but do not change the computation of metagradients. In our practical implementation we do not use the explicit gradient expressions given here. Instead we define a computational graph that includes the inner update and the outer update and compute metagradients via automatic differentiation.

The algorithm updates both the policy parameters η and the meta-parameters ϕ iteratively. In the k-th iteration, it updates the policy parameters η_k using the gradient of the inner objective $J^{\text{inner}}(\eta_k, \tau_k; \phi_k)$, which is parametrized by the meta-parameters ϕ_k. The updated parameters are used for sampling another batch of trajectories τ_{k+1}. The meta-objective $J^{\text{outer}}(\eta_{k+1}, \tau_{k+1})$ is computed as the standard policy gradient loss on the new trajectory. The updated parameters are a differentiable function of the meta-parameters, which enables the computation of metagradients as given in Equation 15. The gradient of the updated parameters

with respect to the meta-parameters can be written as follows

$$\begin{aligned}\nabla_\phi \eta_{k+1} =& \nabla_\phi \Big(\eta_k + \alpha^\eta \nabla_\eta J^{\text{inner}}(\eta_k, \tau_k; \phi_k)\Big) \\ \approx& \alpha^\eta \nabla_\phi \nabla_\eta J^{\text{inner}}(\eta_k, \tau_k; \phi_k) \\ =& \alpha^\eta \Big(\nabla_\phi \nabla_\eta J_\phi(\eta_k, \tau_k) + \beta^{\mathcal{H}} \nabla_\phi \nabla_\eta \mathcal{H}(\pi_{\eta_k})\Big) \\ =& \alpha^\eta \nabla_\phi \nabla_\eta J_\phi(\eta_k, \tau_k)\end{aligned}$$

The second equation is a greedy approximation of the metagradients by dropping the dependency of the parameter η_k on the previous updates (cf. [Xu et al., 2018]). The fourth equation is because the entropy regularization term does not depend on the meta-parameters ϕ. The approximated metagradients can be estimated from sampled trajectories as

$$\nabla_\phi \eta_{k+1} \approx \alpha^\eta \sum_{\tau_k \sim \pi_{\eta_k}} \Big[\sum_{t=0}^{T-1} \nabla_\phi \hat{A}_\phi(s_t, a_t) \nabla_\eta \log \pi_\eta(a_t|s_t)\Big].$$

Dropping the dependency of the parameters on the previous updates introduces bias to the gradient. We leave the investigation of using an unbiased gradient estimator for future work.

A.2 Hardware

We used NVIDIA GeForce GTX 1080 Ti for training the agents in all of our experiments.

A.3 DAG Experiments

Three algorithms were compared in a tabular domain. In this appendix, the hyperparameter configurations of the three algorithms are provided in Section A.3.1 and more detailed results are reported in Section A.3.2.

A.3.1 Hyperparameters

Fixed-λ uses the Adam optimizer [Kingma and Ba, 2014] with learning rate 0.01, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The learning rate was selected from a coarse-grained search in $\{0.1, 0.01, 0.001\}$. The other Adam parameters were chosen by hand without search. Updates are computed on batches consisting of 8 full episodes. We found $\lambda = 1$ the best, better than any smaller values. We used a discount factor $\gamma = 1$ and an entropy regularization coefficient 0.001. The inner loop of Meta-PWTD, Meta-PWR, and H-PWR share hyperparameters with the fixed-λ baseline, except that λ is not used. For Meta-PWTD and Meta-PWR, the outer-loop optimizer is Adam with the same hyperparameters as the inner-loop optimizer. Outer-loop gradient is clipped to 0.5 by global norm. The weight matrix ϕ is initialized from uniform distribution in range $[-0.01, 0.01]$.

A.3.2 Additional Empirical Results

Learning curves in the DAG environment are presented in Figure A.1. All methods converged to the optimal performance by the end of 10000 episodes of training. Handcrafted and meta-learned weights in the depth-4, depth-8, and depth-16 DAG environment variants are presented in Figures A.2, A.3, and A.4.

A.3.3 Learning TD-error Weights with Different Value Functions

In the main chapter we show that the weights learned by Meta-PWR in the η-reset experiment converge to a fixed weight matrix, which resemble the handcrafted weights that we believe are useful for variance reduction (see Figure A.2, Figure A.3, and Figure A.4 for visualizations of the weights). There exist sets of fixed weights that are beneficial throughout the learning process because the reward associated with each transition is stationary. In contrast, the TD-error for each transition is non-stationary and evolves as the policy and the value function change. Accordingly, the weights learned by Meta-PWTD are also non-stationary. To visualize and understand the weights learned by Meta-PWTD, we conducted a set of experiments

where we set the value function by hand and held it fixed during learning so that the weights could converge to a fixed point. Specifically, we use the optimal value function but set the values to zero for all states up to certain depth and hold it fixed during learning.

In Figure A.5, three cases of value masking are shown in the depth-8 DAG environment where the optimal value function has been masked to depth 0, 4, and 8. In the mask depth 0 case, the weights for the TD-errors have mostly changed in the parts of the weight matrix before column 30. Column 30 corresponds to the last state of the middle-layer of the DAG that splits the states where the agent can act from the states where the agent receives rewards. When values are masked up to depth 4, the TD-error weighting shifts. State 14 is the last state at depth 4, so all weights before that are unchanged from the initialization. Note that at mask depths 0 and 4, no weights are placed on the states in the weight matrix after column 30. Finally, at depth 8, all of the value function has been masked out and the TD-error weighting has converged to visually similar weights as the weights learned by Meta-PWR, which is expected as the TD-errors computed with the fully masked value function consist only of the rewards. The learned weights in Figure A.5 show that Meta-PWTD learns different kinds of weightings depending on the value function.

A.4 Key-to-Door Experiments

A.4.1 Environment Description

Key-to-Door (KtD) is a fixed-horizon episodic task where each episode consists of three 2D gridworld phases.

In the *Key phase* (duration 15 steps), there is no reward and the agent must navigate in a 5×5 map to collect a key. The key disappears once collected. The initial locations of the agent and the key are randomly sampled in each episode.

In the *Apple phase* (duration 90 steps), the agent collects apples in a 5×9 map by walking over them; apples disappear once collected. Each apple yields a noisy reward with mean μ and variance σ^2. Specifically, each apple yields a reward of $r = \mu\kappa$ with probability $\frac{1}{\kappa}$ or a reward of 0 with probability $1 - \frac{1}{\kappa}$

where $\kappa = \frac{\sigma^2}{\mu^2} + 1$. This sampling procedure is consistent with the original TVT paper [Hung et al., 2019]. The number of apples is uniformly sampled from $[1, 20]$ and their locations are randomly sampled.

In the *Door phase* (duration 15 steps), the agent starts at the center of a 3×3 room with a door. The agent can open the door only if it has collected the key in the earlier Key phase. The door disappears after being opened. Successfully opening the door yields a reward of 10.

The agent's observation is a tuple, (MAP, HAS_KEY). MAP is the top-down view of the current phase and is rendered in an RGB representation. HAS_KEY is a binary channel which is 1 if the agent has already collected the key and 0 otherwise. The agent has 4 actions which correspond to moving *up*, *down*, *left*, and *right*. The primary difference between our KtD environment and the original is that our environment is fully observable; the original is partially observable. This difference is reflected in two modifications: the agent observes the top-down view of the map rather than the first-person view, and the agent observes whether it has collected the key.

We also conducted experiments in a stochastic variant of KtD to test the robustness of Meta-PWR and Meta-PWTD to stochastic transition dynamics. In the stochastic variant, the action being executed is replaced by a random action with probability 0.1 for each time step.

A.4.2 Implementation Details

All methods use A2C [Mnih et al., 2016] as the policy optimization algorithm. 16 actors are used to generate data. The rollout length is equal to the episode length, 120 in this case. For each method described below, we conducted a hyperparameter search in the $\mu = 5$ and $\sigma = 5$ configuration and selected the best-performing hyperparameters. Then the hyperparameters were fixed for all the other 8 environment configurations. Each candidate hyperparameter combination was run with three different random seeds for 50 million frames. The best hyperparameter combination was determined to be the one that first achieved 57 in episode return,

i.e., 95% of the maximum possible. The following hyperparameter settings are shared across all methods unless otherwise noted: learning rate $2 * 10^{-4}$, and Adam $\beta_1 = 0$, Adam $\beta_2 = 0.999$, Adam $\epsilon = 10^{-8}$, discount factor $\gamma = 0.998$, and entropy regularization coefficient 0.05. The advantage estimates are standardized in a batch of trajectories before computing the policy gradient loss [Dhariwal et al., 2017] unless otherwise noted.

A Standard Perception module. A standard perception module is used by all method to process the observation s to a latent vector h. The observation s is a tuple, (MAP, HAS_KEY). *map* is the top-down view of the current phase that has shape $(7, 11, 3)$, where the last dimension is the RGB channels. HAS_KEY is a binary channel which is 1 if the agent has already collected the key and 0 otherwise. *map* is processed by two convolutional layers with 16 and 32 filters respectively. Both convolutional layers use 3×3 kernels and are followed by ReLU activation. The output of the last convolutional layer is then flattened and processed by Dense(512) - ReLU. The binary input HAS_KEY is concatenated with the ReLU layer output. Finally, the concatenated vector is further processed by a MLP: Dense(512) - ReLU - Dense(256) - ReLU. We denote the final output of the perception module as h.

Fixed-λ The fixed-λ baseline implements the standard A2C algorithm. The policy and value function are implemented by two separate neural networks consisting of a perception module and an output layer without any parameter sharing. The policy network maps h to the policy logits via a single dense layer. The value network maps h to a single scalar via a single dense layer. We label this baseline fixed-λ to underline the importance of the eligibility trace parameter λ. We searched for all combinations of the following hyperparameter sets: λ in $\{1.0, 0.99, 0.98, 0.95, 0.9, 0.8, 0.5, 0\}$, and learning rate in $\{10^{-3}, 2 * 10^{-4}, 5 * 10^{-5}, 10^{-5}\}$. The best performing set of hyperparameters is $\lambda = 0.5$ and learning rate $2 * 10^{-4}$.

Meta-PWTD The policy (η) and value function for the original return (A) have the same network architecture as in the fixed-λ baseline. The meta-network (ϕ) computes the weights as follows. For each episode, the inputs to the meta-network is a sequence $(s_0, \delta_1, s_1, \ldots, \delta_T, s_T)$. Note that the TD-error δ_i is part of the inputs. The meta-network first maps each $s_t (0 \leq t \leq T)$ into a latent vector h_t with a standard perception module. The meta-network (ϕ) shares the perception module with the value function for the original return (A). No gradient is back-propagated from the meta-network to the shared perception module. A dense layer with 256 hidden units maps $h_i (0 \leq i < T)$ into h_i^{row}; a separate dense layer with 256 units maps $h_j \oplus \delta_j (0 < j \leq T)$ into h_j^{col} where $\oplus$ denotes concatenation. The TD-error δ_j is clipped to $[-1, 1]$ before concatenation. Both dense layers are followed by ReLU activation. Another dense layer with 256 units maps the time interval $(j - i)(0 \leq i < j \leq T)$ to a latent vector td_{ij}. h_i^{row}, h_j^{col}, and td_{ij} are element-wise multiplied to fuse the three latent vectors into one vector h_{ij}: $h_{ij} = (h_i^{row} + 1) * (h_j^{col} + 1) * (td_{ij} + 1)$. Note that every vector is shifted by a constant 1 before the multiplication to mitigate gradient vanishing at the beginning of training [Perez et al., 2018]. The latent vectors $h_{ij} (0 \leq i < j \leq T)$ are normalized by

$$h'_{ijd} = \gamma_d \frac{(h_{ijd} - \mu_d)}{\sigma_d} + \beta_d \quad (1 \leq d \leq 256),$$

where

$$\mu_d = \frac{2}{T * (T + 1)} \sum_{0 \leq i < j \leq T} h_{ijd}$$

and

$$\sigma_d = \sqrt{\frac{2}{T * (T + 1)} \sum_{0 \leq i < j \leq T} (h_{ijd} - \mu_d)^2}$$

are the empirical mean and standard deviation of h_{ij} respectively. γ_d and β_d are trainable parameters. ReLU activation is applied to h'_{ij}. Finally, the output layer maps each h'_{ij} into w_{ij}. The initial weights for the output layer is scaled by a factor 0.01 so that the initial outputs are closer to uniform. We applied sigmoid activation on the outputs to bound the weights to $(0, 1)$. As for hyperparameters, the entropy regularization coefficient is set to 0.05. For the inner loop, we used the

Adam optimizer with learning rate $2 * 10^{-4}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. For the outer loop, we used the Adam optimizer with learning rate $2 * 10^{-5}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Meta-PWR Meta-PWR uses the same neural network architecture as Meta-PWTD. The only difference is that Meta-PWR does *not* take the reward r_i or the TD-error δ_i as inputs. In addition, Meta-PWR employs a value function for the weighted sum of rewards (ϕ) which has an identical architecture as the value function for the original return (A). The hyperparameters for Meta-PWR are also the same as those for Meta-PWTD.

H-PWR The policy and value function for the weighted sum of rewards have the same network architecture as in the fixed-λ baseline. We handcraft pairwise weights for the KtD domain to take advantage of the known credit assignment structure that can be described as follows: The policy learning in the key phase depends only on the reward in the door phase, the policy learning in the apple phase does not depend on the other phases, and while the picking up the key or not impacts the reward in the door phase, the reward in the door phase is still instantaneous. The weights are set so that in the key phase, the rewards in the apple phase receive a zero weight and the rewards in the door phase are discounted starting from the first timestep of the door phase. Weights that compute discounting equivalent to $\gamma = 0.92$ are applied in the apple phase and door phase. An illustration of the learned weights is presented in Figure 3 in the chapter 4. H-PWR uses the same hyperparameters as fixed-λ except γ, which does not apply and λ, which is set to 1.0. No hyperparameter search is conducted specifically for H-PWR.

Meta-$\lambda(s)$ The policy (η) and value function for the original return (A) have the same network architecture as in the fixed-λ baseline. The value function for the weighted sum of rewards (ϕ) has identical architecture as the value function for the original return (A). The meta-network (ϕ) maps a state s_t to a scalar $\lambda(s_t) \in (0, 1)$. The meta-network first maps the observation s_t to a latent vector

h_t with the standard perception module and then maps h_t to a single scalar $\lambda(s_t)$ via a single dense layer. Sigmoid is applied to the output of the meta-network to bound it to $(0, 1)$. We searched for the outer-loop learning rate in $\{10^{-4}, 2 * 10^{-5}, 5 * 10^{-6}, 10^{-6}\}$. The best performing outer-loop learning rate is 10^{-6}. The outer-loop λ is set to 1.0 without search.

RGM The policy (η) and value function for the original return (A) have the same network architecture as in the fixed-λ baseline. The value function for the weighted sum of rewards (ϕ) has identical architecture as the value function for the original return (A). For an episode $\tau = (s_0, a_0, r_1, s_1, \ldots, s_T)$, the meta-network first maps each s_t $(0 \leq t \leq T)$ to h_t^0 by a shared standard perception module. Then it concatenates h_t^0 with r_t and the one-hot representation of a_t. Four Transformer blocks [Vaswani et al., 2017] are applied on the concatenated features, each block with four attention heads. We denote the output of the final Transformer block as h_t^4 $(0 \leq t \leq T)$. Finally, a shared linear layer maps each h_t^4 to β_t, the weight on the reward r_t. We searched for the outer-loop learning rate in $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$, as suggested by the original paper [Wang et al., 2019b]. The best performing outer-loop learning rate is 10^{-6}. The outer-loop λ is set to 1.0 without search.

TVT We adopted the agent architecture from the open-source implementation accompanying the original TVT paper (https://github.com/deepmind/deepmind-research/tree/master/tvt). The only difference is that we replaced the convolutional neural network torso in the original code with the standard perception module. We searched for all combinations of the following hyperparameter sets: the read strength threshold for triggering the splice event in $\{1, 2\}$ and the learning rate in $\{10^{-3}, 2 * 10^{-4}, 5 * 10^{-5}, 10^{-5}\}$. The best performing set of hyperparameters are 1 for the read strength threshold and $2 * 10^{-4}$ for the learning rate. We did not use advantage standardization for TVT because we found that it hurt the performance in the KtD domain. We used the Adam optimizer parameters $\beta_1 = 0$, $\beta_2 = 0.95$, and $\epsilon = 10^{-6}$, as the open-source implementation suggested. We also set $\lambda = 0.92$ and $\gamma = 0.92$ following the original implementation.

A2C-RR We pick the main ideas from RUDDER [Arjona-Medina et al., 2019] and implement them in an algorithm we call A2C-RR. RUDDER uses contribution analysis to redistribute rewards in a RL episode. The high-level idea is that since the environment rewards may be delayed from the transitions that resulted in them, contribution analysis may be used to compute how much of the total return is explained by any particular transition. In effect, this drives the expected future return to zero because any reward that can be expected at any given timestep will be included in the redistributed reward, eliminating the delay and leading to faster learning of the RL agent. The method is based on learning a LSTM-network, which predicts the total episodic return at every timestep. The redistributed reward is computed as the difference of return predictions on consecutive timesteps.

Compared to the full RUDDER algorithm, A2C-RR incorporates a few changes to isolate some of the core ideas of the reward redistribution module and make the algorithm more directly comparable to the other A2C-based algorithms discussed in chapter 4. We use the LSTM cell-architecture proposed in the RUDDER paper and train it from samples stored in a replay buffer. In the KtD-experiments, we do not use the "quality" weighted advantage estimate proposed in the paper due to the random noise in the episodic return, which we deem too high variance for reliable estimation of the redistribution quality. Instead we mix the original and redistribution-based advantages at a fixed ratio. We recognize that omitting some of the features of the full RUDDER algorithm may adversely impact the reward redistribution and therefore the agent learning performance. Nevertheless, we believe the reward redistribution idea is an interesting take on a similar idea as the pairwise weighting studied in chapter 4 and therefore provide our implementation – A2C-RR – of that idea as a baseline.

The regular frames and delta frames (as described in chapter 4) are processed by the standard perception module. The preception module outputs and the one-hot encoded action are concatenated and processed by Dense(512) - ReLU. The output of the ReLU layer is the input for the reward redistribution model. As suggested in the paper, the reward redistribution model is a LSTM without a forget gate

and output gate. The cell input only receives forward connections and the gates only receive recurrent connections. All of the layers in the LSTM have 64 units. We chose not to use the prioritized replay buffer described in the paper due to the high variance of the returns in the KtD environment. For the same reason we did not use the quality measure, which is also described in the paper, for mixing the RUDDER advantage and regular advantage. Instead, we used a fixed mixing coefficient, which we searched for. The advantage is standardized after the mixing. We implemented an auxiliary task described in the paper, where the total return prediction loss is applied at every step of the episode. The reward redistribution model is trained for 10 randomly sampled batches of size 8 from a circular buffer holding the past 128 trajectories between each policy update. We set the number of updates to 10 via an informal hyperparameter search in the $\mu = 5$, $\sigma = 5$ KtD setting, where we found that training 10 times between each update performs better than 5 but further increasing it did not yield further large improvements. The reward redistribution model is trained with Adam with learning rate 10^{-4}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We applied a L2 weight regularizer with coefficient 10^{-7}. We searched for all combinations of the following hyperparameter sets: γ in $(0.92, 1.0)$, λ in $(1.0, 0.95, 0.5)$, auxiliary task coefficient in $(0.0, 0.5)$, and advantage mixing coefficient in $(0.5, 1.0)$. The best performing set of hyperparameters is $\gamma = 0.92$, $\lambda = 0.5$, auxiliary task coefficient 0.0, and advantage mixing coefficient 0.5.

A.4.3 Additional Empirical Results

We ran all of the methods described above in 9 variants of the KtD environment. Figure A.6, Figure A.7, and Figure A.8 shows the episode return, the total reward in the door phase, and the total reward in the apple phase respectively.

Noticing that the KtD domain has deterministic dynamics, we also conducted experiments on a stochastic KtD domain where the action being executed is replaced by a random action with probability 0.1 for each time step. The corresponding

results are presented in Figure A.9, Figure A.10, and Figure A.11. In general, Meta-PWTD and Meta-PWR still perform better than the baseline methods regardless of the stochasticity in the transition dynamics.

A.5 `bsuite` Experiments

A.5.1 Environment Description

We selected 7 tasks which were associated with the "credit assignment" tag from `bsuite`. They present a variety of credit assignment structures, including the umbrella problem in §3 in the main text. Additionally, all domains except *Discount Chain* have multiple tags which create additional challenges than temporal credit assignment. We ran all different variants of every task, each with 3 different random seeds. Unlike the standard data regime of `bsuite`, we ran each task for $100K$ episodes for all methods to calculate the total regret score, except *Cartpole*, which we ran for $50K$ episodes. We refer the readers to the original `bsuite` paper [Osband et al., 2019] and the accompanying github repository (https://github.com/deepmind/bsuite) for further details.

A.5.2 Implementation Details

Most methods use a similar neural network architecture as described in §A.4.2. There are two common differences. First, we used 1 single actor instead of 16 parallel actors for generating data. Second, the standard perception module is replaced by a 2-layer MLP with 64 hidden units and ReLU activation each layer, because the inputs are vectors instead of images in `bsuite`. Further architecture differences and hyperparameters are described below.

Actor-critic Baseline The entropy regularization weight is set to 0.05. we used the Adam optimizer with learning rate $3 * 10^{-4}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Meta-PWTD and Meta-PWR Besides the perception module, there are two more differences with §A.4.2. First, the meta-network (ϕ) and the value function for the original return (A) use separate perception module instead of sharing. Second, all the hidden layers after the perception module use 64 hidden units instead of 256. The entropy regularization weight is set to 0.05. For the inner loop, we used the Adam optimizer with learning rate $3 * 10^{-4}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. For the outer loop, we used the Adam optimizer with learning rate $3 * 10^{-5}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. Outer-loop gradient is clipped to 0.01 by global norm.

RGM The entropy regularization weight is set to 0.05. For the inner loop, we used the Adam optimizer with learning rate $3 * 10^{-4}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. For the outer loop, we used the Adam optimizer with learning rate $1 * 10^{-4}$, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

A2C-RR Apart from the perception module, the same A2C-RR implementation was used for `bsuite` as was used for KtD experiments. The actor-critic was trained with the same hyperparameters as the Actor-Critic baseline for `bsuite`. The LSTM was trained with the same hyperparameters as in KtD.

A.6 Atari Experiment

A.6.1 Implementation Details

Most methods use a similar neural network architecture as described in §A.4.2. There are two common differences. First, we generate 20-step trajectories instead of full episodes for each policy update, following the original A2C implementation [Mnih et al., 2016]. Second, the standard perception module is replaced by the convolutional neural network architecture used in [Mnih et al., 2015]. Further architecture differences and hyperparameters are described below.

A2C The policy and the value function share the perception module. The value loss coefficient is 0.5. The entropy regularization coefficient is 0.01. We used the RMSProp optimizer with learning rate 0.0007, decay 0.99, and $\epsilon = 10^{-5}$. The gradient is clipped to 0.5 by global norm. The discount factor γ is 0.99. We searched for the eligibility traces parameter λ in $\{0.8, 0.9, 0.95, 0.98, 0.99, 1\}$ and selected 0.95.

Meta-PWTD The inner-loop hyperparameters are exactly the same as the A2C baseline. For the outer loop, We applied an entropy regularization as well to stabilize training. The coefficient is 0.01. The outer loop uses the Adam optimizer with learning rate 0.00003, $\beta_1 = 0$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The outer-loop gradient is clipped to 0.05 by global norm.

A2C-RR To handle the variable length episodes in Atari, we chunk the trajectories as described in the RUDDER paper [Arjona-Medina et al., 2019]. Unlike RUDDER, A2C-RR uses a circular replay buffer, to which all trajectories are added and samples training batches from the buffer uniformly. We use the quality measure described in the RUDDER paper for mixing the advantage estimates to the A2C-RR implementation for Atari. The quality is computed as described in the paper, and used for mixing the advantage computed with the environment rewards and the one computed with the redistributed rewards. The quality is also used as the coefficient for the mean-squared error loss used for training the baseline for the redistributed reward. The LSTM is trained for a maximum of 100 LSTM epochs every 100 actor-critic training iterations. If the quality of the last 40 LSTM training trajectories is positive after updating the LSTM, the LSTM training is stopped. For training the LSTM, we normalize the rewards by the maximum return encountered so far and multiply the normalized rewards by 10. Before mixing the regular and the redistributed advantages, we denormalize the redistributed rewards by inverting the normalization process above. The inputs to the LSTM are the delta-frames and one-hot encoded actions. In the RUDDER paper, a more sophisticated exploration strategy is used for collecting data. We did not implement it for a fair comparison to other methods.

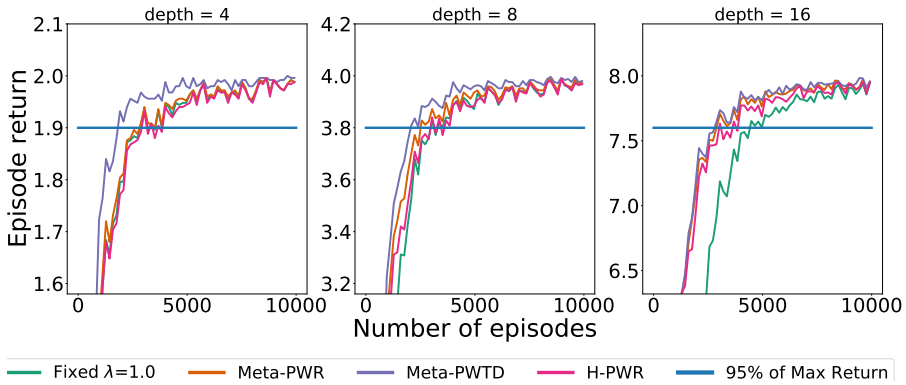


Figure A.1: Learning curves in the DAG environments. Each curve is the mean of 5 seeds. The line for 95% of max return is added to help contextualize these learning curves with Figure 2 in chapter 4.

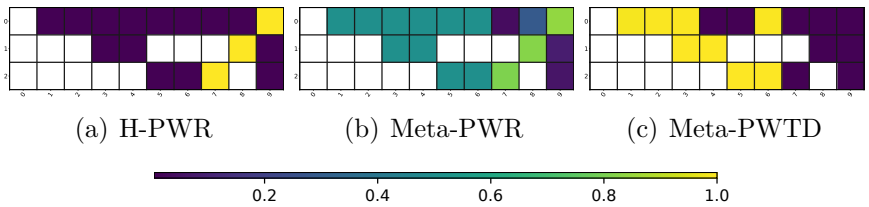
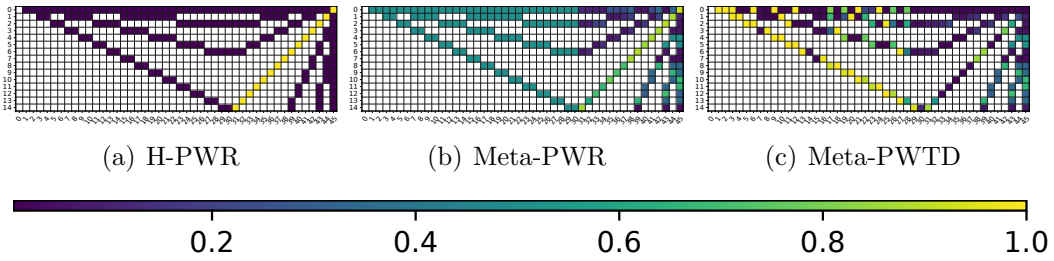


Figure A.2: Handcrafted, meta-learned reward weights, and meta-learned TD-error weights in the depth-4 DAG environment in **a**, **b**, and **c** respectively. White is unreachable. Y-axis has been cropped to only include weights that affect inner loop learning.

A2C-RR uses the same A2C agent as the Fixed-λ baseline, with the same hyperparameters. Apart from the differences described above, the LSTM training hyperparameters are from the RUDDER paper [Arjona-Medina et al., 2019]. The LSTM is trained with trajectories of length 512, further split into chunks of length 128. The LSTM training batch size is 8, learning rate is 10^{-4}, the optimizer is ADAM with $\epsilon = 10^{-8}$, gradient clip is 0.5, and the L_2-regularizer coefficient is 10^{-7}. The LSTM starts training after at least 32 trajectories have been collected from the environment. The replay buffer stores maximum of 128 trajectories of length 512.

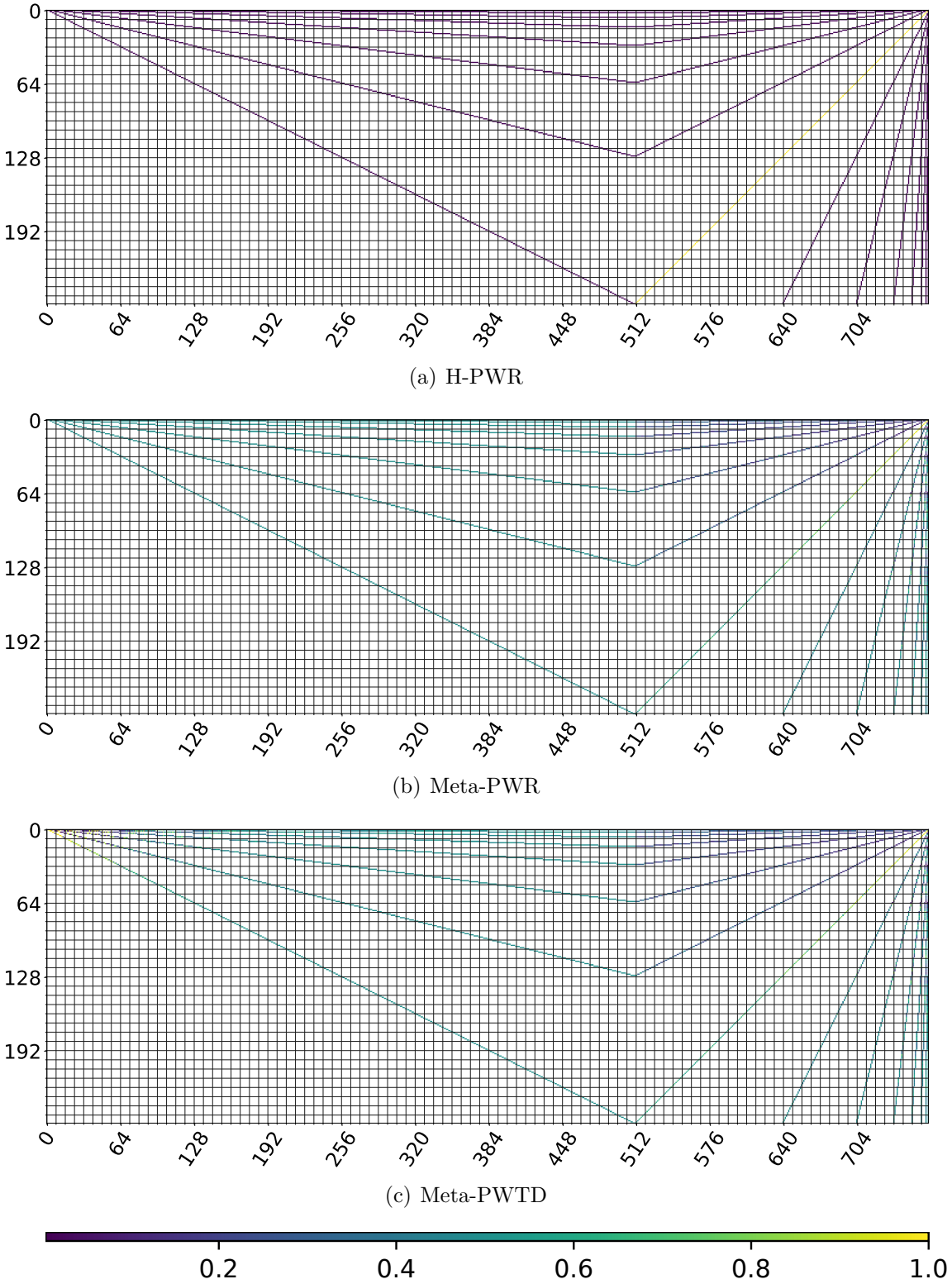
A.6.2 Additional Empirical Results

Figure A.12 shows the learning curves of Meta-PWTD, A2C-RR, and the A2C baseline.



(a) H-PWR (b) Meta-PWR (c) Meta-PWTD

Figure A.3: Handcrafted, meta-learned reward weights, and meta-learned TD-error weights in the depth-8 DAG environment in **a**, **b**, and **c** respectively. White is unreachable. Y-axis has been cropped to only include weights that affect inner loop learning.

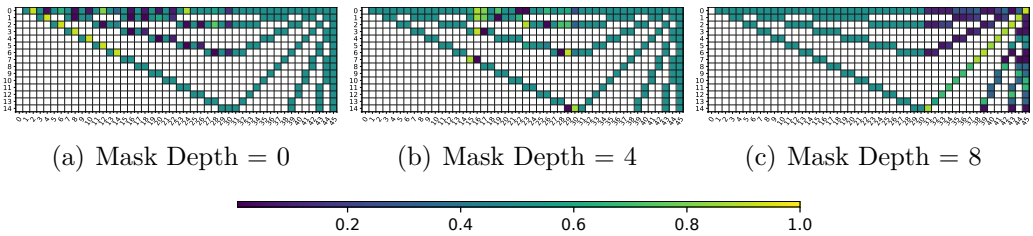


(a) H-PWR

(b) Meta-PWR

(c) Meta-PWTD

Figure A.4: Handcrafted, meta-learned reward weights, and meta-learned TD-error weights in the depth-16 DAG environment in **a**, **b**, and **c** respectively. White is unreachable. Y-axis has been cropped to only include weights that affect inner loop learning.



(a) Mask Depth = 0 (b) Mask Depth = 4 (c) Mask Depth = 8

Figure A.5: Meta-PWTD inner loop-reset weight visualization experiment with masked optimal value function. Figures **(a,b,c)** show the learned weight matrices for mask depths 0, 4, and 8 respectively.

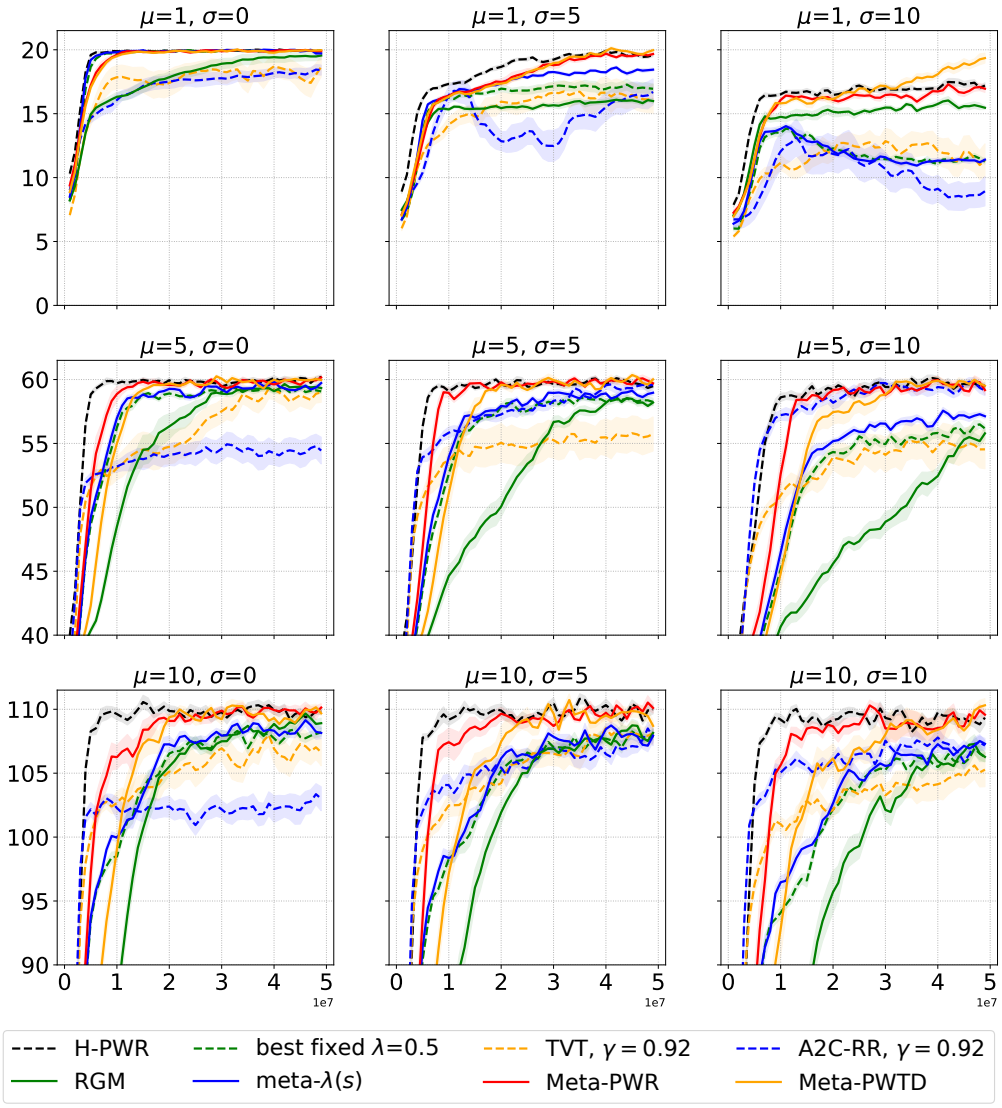


Figure A.6: Episode returns in all 9 variants of the Key-to-Door environment. The x-axis is the number of frames. Rows are the different apple reward means (μ), columns the different apple reward variance (σ^2). The x-axis reflects the number of frames, y-axis the episode return. The curves show the average over 10 independent runs with different random seeds and the shaded area shows the standard errors.

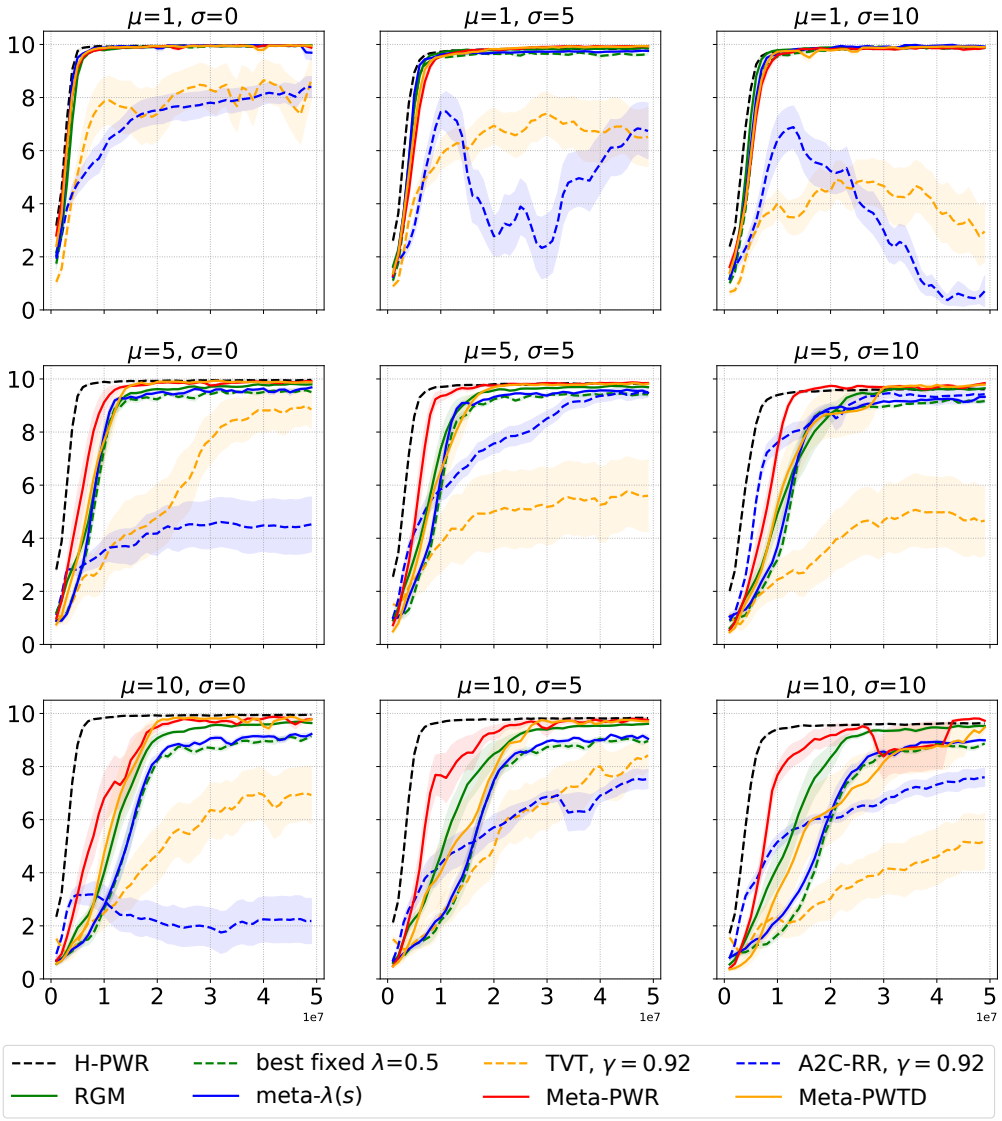


Figure A.7: Door phase rewards in all 9 variants of the Key-to-Door environment. The x-axis is the number of frames. Rows are the different apple reward means (μ), columns the different apple reward variance (σ^2). The x-axis reflects the number of frames, y-axis the door phase return. The curves show the average over 10 independent runs with different random seeds and the shaded area shows the standard errors.

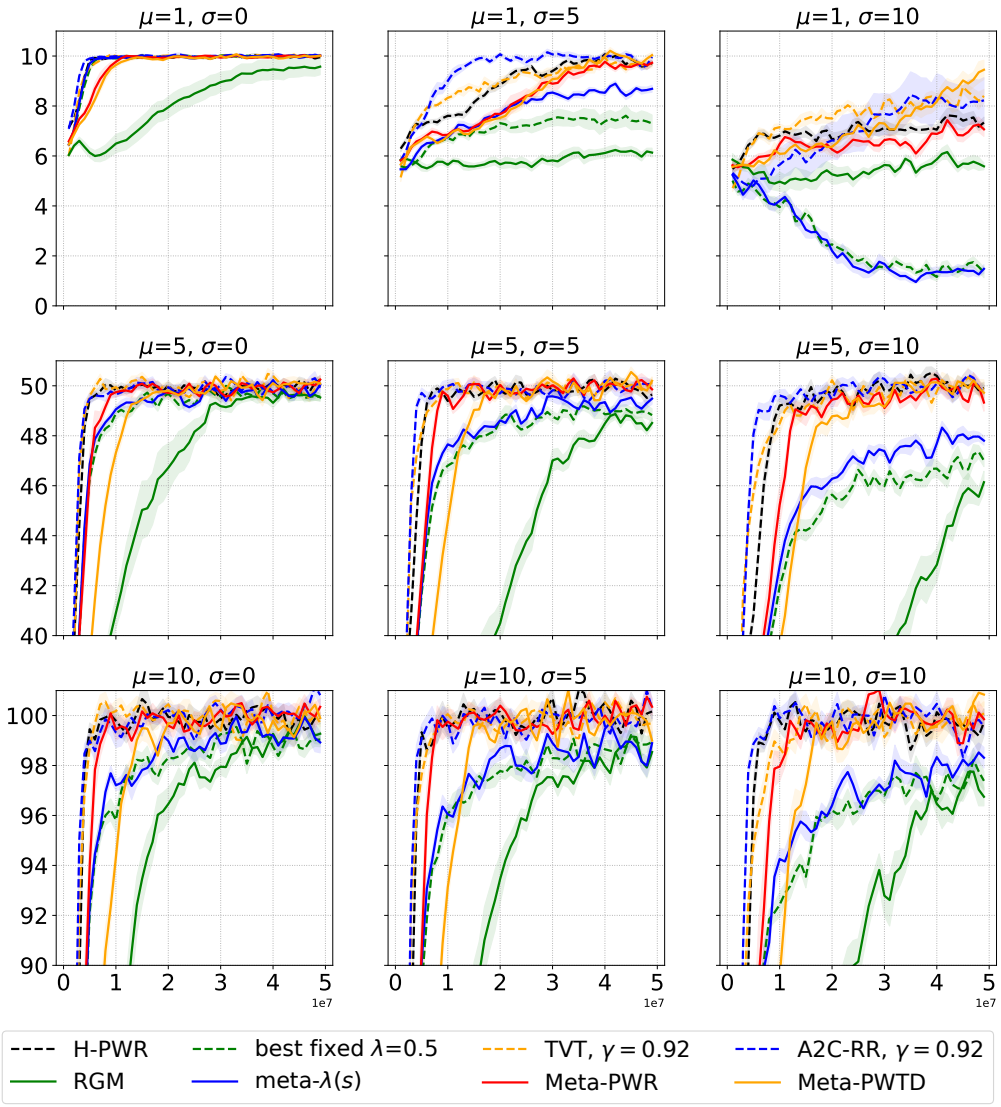


Figure A.8: Apple phase rewards in all 9 variants of the Key-to-Door environment. The x-axis is the number of frames. Rows are the different apple reward means (μ), columns the different apple reward variance (σ^2). The x-axis reflects the number of frames, y-axis the apple phase return. The curves show the average over 10 independent runs with different random seeds and the shaded area shows the standard errors.

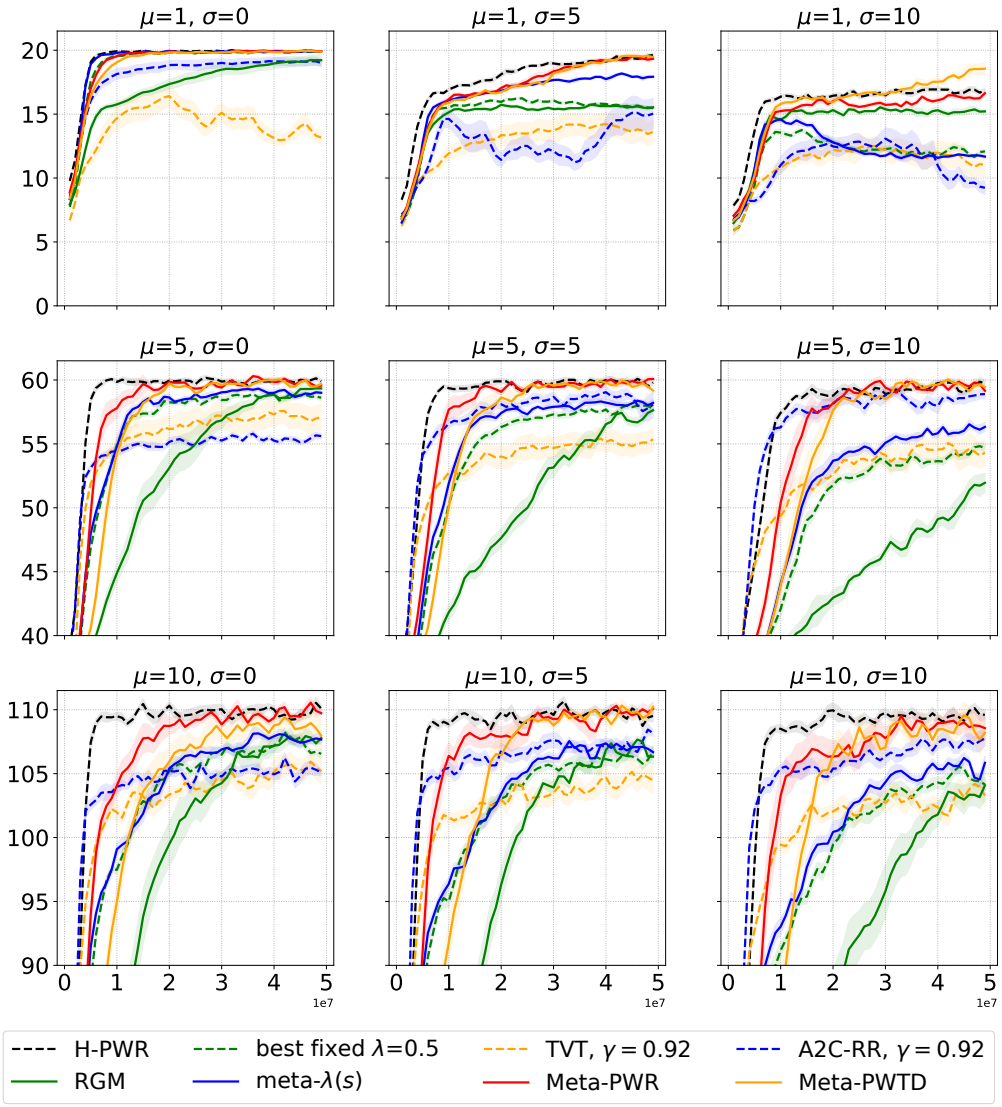


Figure A.9: Episode returns in all 9 variants of the stochastic Key-to-Door environment. The x-axis is the number of frames. Rows are the different apple reward means (μ), columns the different apple reward variance (σ^2). The x-axis reflects the number of frames, y-axis the episode return. The curves show the average over 10 independent runs with different random seeds and the shaded area shows the standard errors.

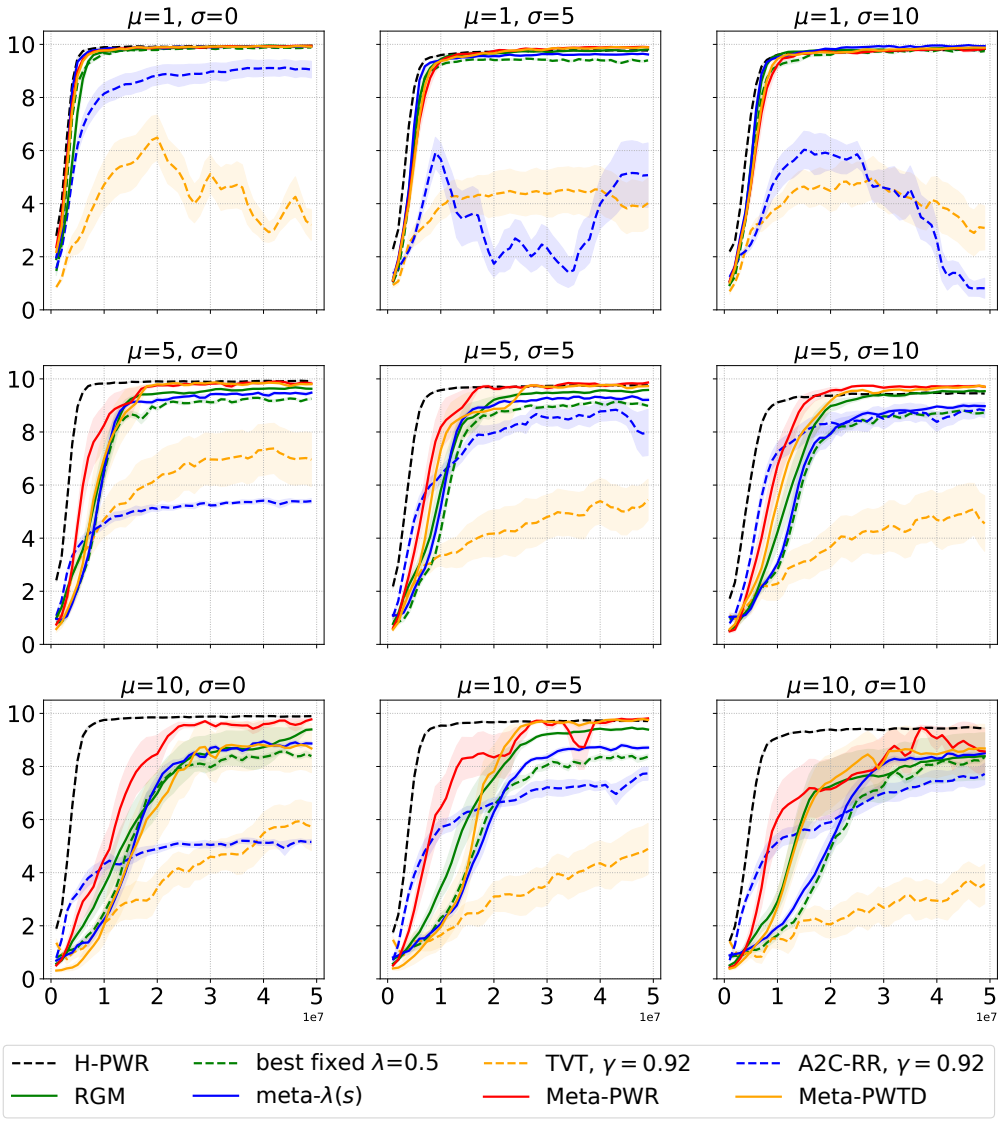


Figure A.10: Door phase rewards in all 9 variants of the stochastic Key-to-Door environment. The x-axis is the number of frames. Rows are the different apple reward means (μ), columns the different apple reward variance (σ^2). The x-axis reflects the number of frames, y-axis the door phase return. The curves show the average over 10 independent runs with different random seeds and the shaded area shows the standard errors.

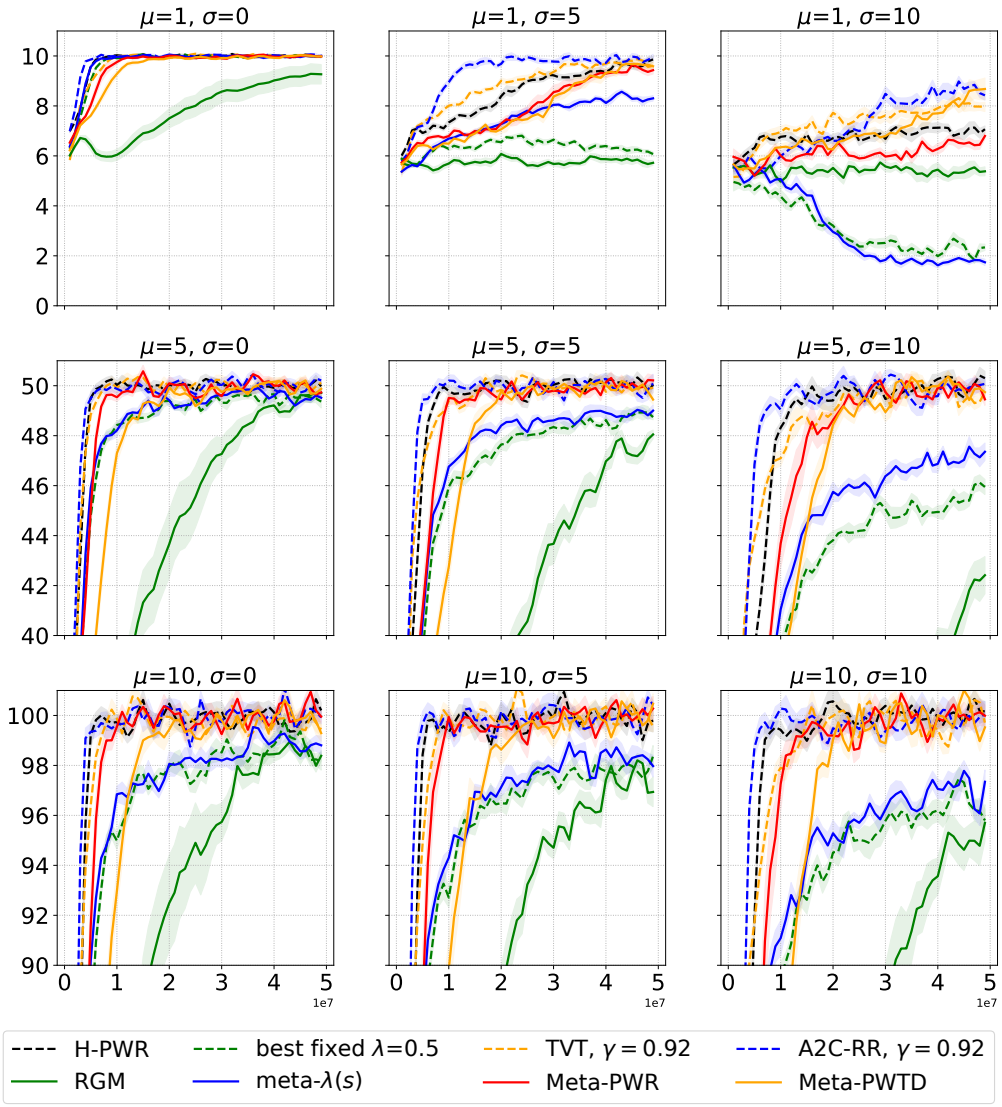


Figure A.11: Apple phase rewards in all 9 variants of the stochastic Key-to-Door environment. The x-axis is the number of frames. Rows are the different apple reward means (μ), columns the different apple reward variance (σ^2). The x-axis reflects the number of frames, y-axis the apple phase return. The curves show the average over 10 independent runs with different random seeds and the shaded area shows the standard errors.

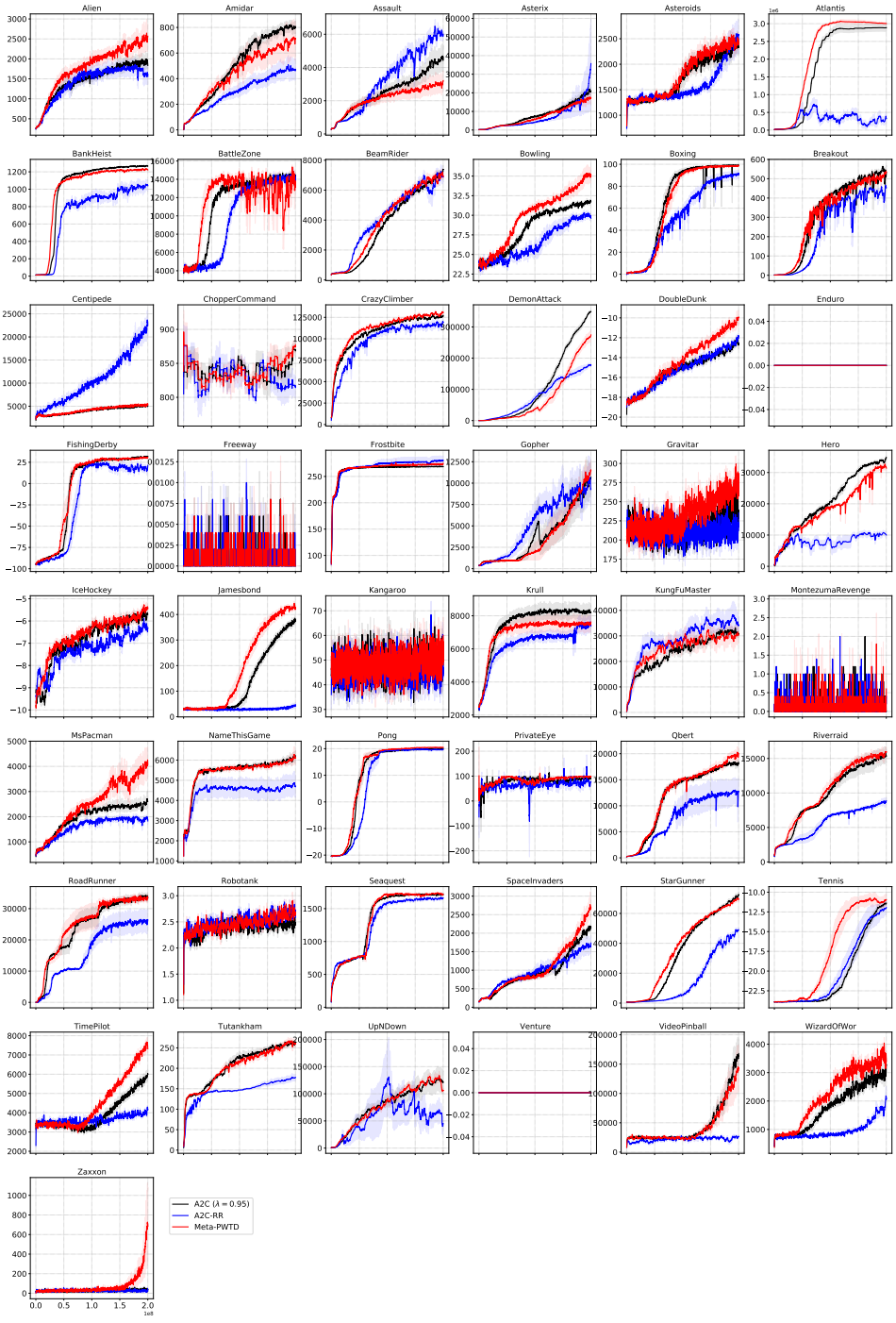


Figure A.12: Learning curves on 49 Atari games. The x-axis is the number of frames and the y-axis is the episode return. Each curve is averaged over 5 independent runs with different random seeds. Shaded area shows the standard error over 5 runs.

B

An Investigation of the Bias-Variance Tradeoff in Meta-Gradients

Contents

B.1 Meta-Gradient Estimators for Expected Policy Gradients

An expression for the meta-gradient without the sampling correction, used by most meta-gradient algorithms, can be derived by substituting the stochastic policy gradient in the update function with the expected policy gradient given by equation 2.9. In that case, there is no dependency between the policy η^k and the data sampled with earlier policies because when the policy is updated using the expected policy gradient, η^k becomes a deterministic function of ϕ and η^0. Therefore,

we no longer get the sampling correction terms that consider how ϕ affects the distribution of data. The meta-gradient for the expected update case is given by

$$\nabla\phi J'_K(\phi) = \nabla_\phi \sum_{k=0}^{K} \mathbb{E}_{\tau\sim p(\tau;\eta^k)}\Big[R(\tau)\Big] = \sum_{k=0}^{K} \mathbb{E}_{\tau\sim p(\tau;\eta^k)}\Big[\nabla_\phi \eta^k \nabla_{\eta^k} \log p(\tau;\eta^k) R(\tau)\Big]. \tag{B.1}$$

Algorithms that estimate this by computing inner-loop updates and the meta-gradient from samples are biased because they ignore the sampling correction terms.

B.2 Exponential Discounting of Sampling Corrections for Variance Reduction

We consider an alternative weighting using exponential discounting motivated by analogy to the usual discounting procedure in RL. The exponentially discounted meta-gradient estimator is expressed as follows

$$\sum_{k=0}^{K} \frac{1}{|\mathcal{D}|}\left(\sum_{j=0}^{k-1} \alpha^{k-j} \nabla_\phi \eta^j \nabla_{\eta^j} \log p(\mathcal{D}^j;\eta^j) \sum_{\tau\in\mathcal{D}^k} R(\tau) + \sum_{\tau\in\mathcal{D}^k} \nabla_\phi \eta^k \nabla_{\eta^k} \log p(\tau;\eta^k) R(\tau)\right), \tag{B.2}$$

where α is a meta-discount factor. Analogously to discounting in RL, the weights on the sampling correction terms become exponentially smaller the further back along the update trajectory they are from the return at k. Unlike in standard RL, the discount on the sampling correction term uniformly weights all of the terms in a batch, because all the data in the batch arrives to the update $U(\phi, \eta, \mathcal{D})$ simultaneously.

We compare this variance reduction scheme with the uniform weighting experimentally. In figure B.5, we show that the exponential discounting of the sampling correction terms results in a similar bias-variance tradeoff to the simpler sampling correction coefficient. Therefore, at least in this bandit setting, exponential discounting does not seem to have an advantage over using the uniform weighting.

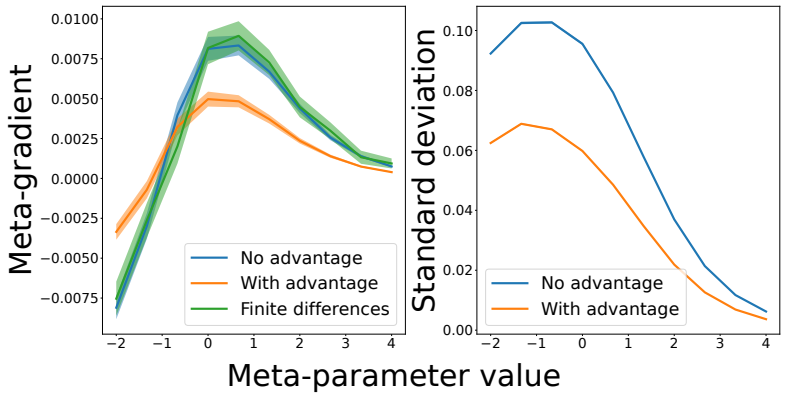


Figure B.1: Comparing meta-gradient estimators with and without advantage estimation with a ground truth computed by finite differences. The x-axis is the value of the entropy coefficient being tuned with meta-gradients. In the left panel, the meta-gradient is computed at different meta-parameter values. The shading shows 95% confidence interval of the mean. In the right panel, the standard deviation of the meta-gradient is computed at different meta-parameter values. The bootstrap estimate of the 95% CI for the standard deviation estimate is sufficiently small that it cannot be rendered in this figure.

B.3 Advantage Estimation with Sampling Corrections

In the previous experiment, the variance of the meta-gradient was mitigated by using a large batch size to estimate the meta-gradient. In more complex RL problems, variance reduction via algorithmic means is important due to limited computation budgets making it difficult to use large batch sizes. In this experiment, we demonstrate that using a standard advantage estimator with the sampling correction results in bias and illustrate the resulting bias-variance tradeoff. This experiment considers a similar problem setting as the previous one with the main differences that the nonstationarity is removed by only considering a single truncation window starting from the initial parameters and that a single scalar entropy coefficient is tuned with the meta-gradient. Sigmoid activation is applied to the entropy coefficient. Details and hyperparameters of the experiment are provided in appendix B.5.

Figure B.1 shows the bias and variance of the untruncated, sampling corrected meta-gradient estimator with and without advantage estimation. The bias is computed as the Euclidean distance to a finite differences estimate of the meta-gradient. The comparison verifies the results in section 5.3.4: using the advantage

estimator with the sampling correction does reduce variance, but unlike with standard policy gradients, it introduces bias. In practice, whether trading off variance for bias is practical depends on the problem setting. In our experiments, we chose not to use the advantage estimators to focus on the bias from truncation and sampling correction, but using the advantage estimator may be worthwhile in practice. In a practical meta-gradient algorithm using sampling corrections, variance reduction is crucial for performance, so research into unbiased advantage estimators in this context is a promising direction for future work.

B.4 Bandit Settings

In this section we describe the details of the bandit experiments. The experiments are conducted in a multi-armed bandit setting. The bandits have 30 arms with the arm means sampled from $exp(Uniform(-100, 1))$. The reward for each pull is computed by adding noise sampled from Gaussian with standard deviation 2 to the arm mean. The inner learning problem is learning the policy for the bandit and the outer problem is to learn a learning rate schedule for the inner learner. The learning rates are grouped into two buckets to produce a two-dimensional meta-parameter for the ease of visualization. The first eight learning rates are grouped into one bucket and the remaining twenty one into another, choosing an uneven split because the first few updates have larger impact on the final performance than the later ones.

The policy is parametrized as a softmax over a randomly initialized vector of logits. The inner loop uses a simple policy gradient algorithm [Williams, 1992]. The inner loop updates the policy for 29 update steps sampling 30 batches of experience in total. Each update in the inner loop is computed on ten samples from the bandit. The outer loop updates the learning rate schedule with the meta-gradient computed by equation 5.3. When truncation is applied, the inner loop is run as before but the outer loop only considers returns within the truncation window and only backpropagates gradients within the window. The meta-gradient is computed across multiple inner learning problems in parallel. For computing one meta-gradient update with the untruncated inner loop, inner batch size $\times$ lifetime length $\times$

	Hyperparameter	Value
Shared hyperparameters	inner batch size	10
	optimizer	SGD
Figure 5.1 left panel	parallel runs	1000
	lifetime length	30
	outer loop updates	100000
Figure 5.2 right panel and figure B.4	parallel runs	1000
	outer loop updates	100000
	outer learning rate	0.01
	lifetime length	80
	number of random seeds	10
Figure 5.1 right panel	parallel runs	1000
	outer loop updates	200000
	outer learning rate	0.001
	number of random seeds	5
	lifetime length	30
	optimizer	ADAM
Figure B.2, truncation lengths 1 and 8	parallel runs	1000
	outer loop updates	500000
	outer learning rate	0.05
	lifetime length	30
	number of random seeds	5
Figure B.2, truncation lengths 22 and 29	parallel runs	1000
	outer loop updates	500000
	outer learning rate	0.005
	lifetime length	30
	number of random seeds	5
Figure 5.2 left panel	parallel runs	2000
	lifetime length	30
	outer loop samples	1000

Table B.1: Bandit hyperparameters for each experiment

parallel runs samples from the bandit are used. All the updates are computed with stochastic gradient descent. The hyperparameters of the experiments in each figure are summarized in Table B.1.

B.5 MDP Settings

In this section we describe the details of the gridworld experiments. This experiment is inspired by the experiment in Flennerhag et al. [2021], but differs in details which are explained below. The experiments are conducted in gridworld environment,

which consists of a 5×5 room. The room has two objects. The task of the agent is to learn to navigate to the rewarding object repeatedly. The positions of the agent and the objects are randomized in the beginning of the episode. Each time the agent reaches one of the objects, the locations of the objects are randomized. The environment is episodic with episode length capped at 16 to avoid having to use value function for bootstrapping. The agent receives a reward of 1 for moving to a cell with one object and -1 for the other object. Every timestep when the agent does not hit a rewarding object, it gets a reward of -0.04. The action space of the agent consists of the cardinal directions. When the agent hits the edge of the room, it stays in the same grid cell. The observation space is the coordinates of the agent and the two objects represented as one-hot vectors, and an additional one-hot vector representing the timestep in the episode to make the environment fully observable. The environment is made non-stationary by flipping the rewards of the objects every 6400 timesteps.

The agent is trained using a simple policy gradient algorithm with a baseline trained alongside the agent and an entropy regularization term, which has its coefficient predicted by the meta-learner. The agent is parametrized as an MLP with two layers of 256 units and ReLU activations. The value function is approximated by another MLP with the same architecture. The value function is only used in the inner-loop. The inner-loop computes updates over five environments in parallel and samples a full episode of 16 steps from each of the environments for each update. The parameters of the policy and value function are updated with SGD. The meta-learning problem is to learn a small MLP, which outputs the coefficient for the entropy regularization term used by the inner-loop. The meta-learner is parametrized as an MLP with a single hidden layer of 32 units with ReLU activation. The output activation is sigmoid. The input to the MLP is a vector of mean rewards in the previous 10 batches used for the inner loop update. The meta-learner is updated by the meta-gradient given by equation 5.3 computed as an average across 50 independent inner-loop learners running in parallel. ADAM optimizer is used for the meta-learning. Hyperparameters of the inner and outer-loops are given in table B.2.

	Hyperparameter	Value
Shared hyperparameters	inner learning rate	1.0
	inner value loss coefficient	0.1
	inner optimizer	SGD
	discount rate γ	0.99
Hyperparameters for figure 5.3	inner batch size	5
	outer batch size	50
	outer optimizer	ADAM
	sampling horizon	16
	maze size	5×5
	reward flip interval	6400
	sampling correction coefficient	$\{0.0, 1.0\}$
	truncation length K	$\{8, 16, 32, 64\}$
	outer learning rate	$\{1e-6, 5e-6\}$
	number of random seeds	3
	ES optimizer	ADAM as above
	ES other hyperparameters	default evosax OpenES
Hyperparameters for figure B.1	inner batch size	10
	outer batch size	25
	outer optimizer	N/A
	sampling horizon	8
	maze size	3×3
	reward flip interval	64
	sampling correction coefficient	1.0
	truncation length K	8
	outer learning rate	N/A
	number of meta-gradient samples	62500

Table B.2: MDP hyperparameters for each experiment

In section B.3 we compare the bias and variance of the sampling corrected, untruncated estimator with and without advantage estimation. The experiment is run in a maze setting similar to the previous one. The agent parameters η are reset at every reward flip interval, removing the nonstationarity from the point of view of the inner-loop. The hyperparameters for the bias variance experiment are given in table B.2.

B.6 Source Code Release

The source code for reproducing the results in this chapter is provided at `https://github.com/vuoristo/meta-gradients`.

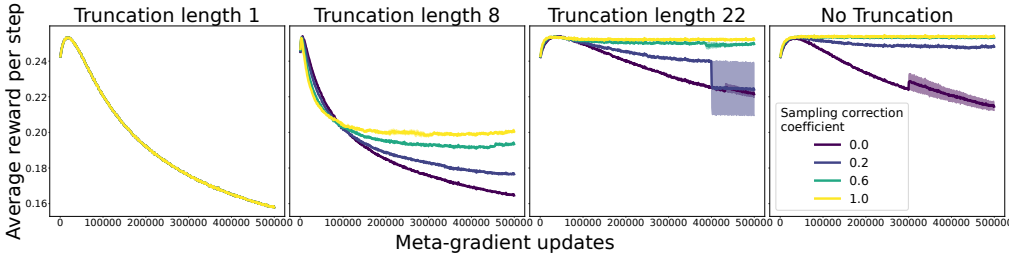


Figure B.2: Learning curves for the meta-gradient estimators in the bandit setting with truncated meta-optimization horizons using SGD as the optimizer. The shading of the curves shows the standard error across random seeds.

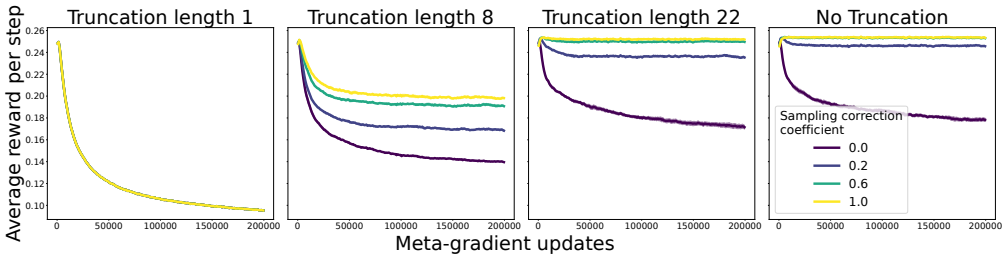


Figure B.3: Learning curves for the meta-gradient estimators in the bandit setting with truncated meta-optimization horizons using ADAM as the optimizer. The shading of the curves shows the standard error across random seeds.

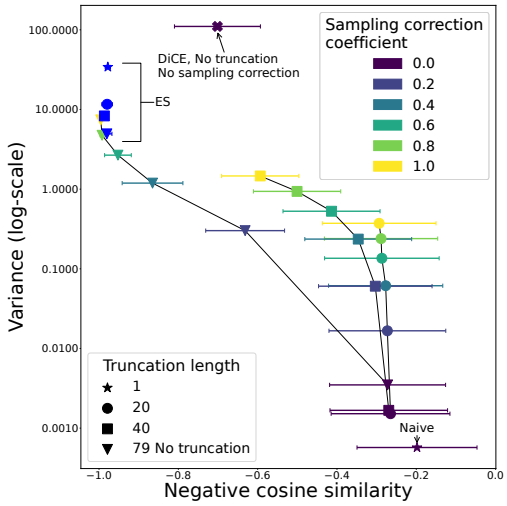


Figure B.4: The bias and variance of meta-gradient estimators in the bandit setting with negative cosine similarity estimate of the bias. Negative cosine similarity is shown instead of cosine similarity to keep the bias axis consistent with figure 5.2, that is, bias grows toward the right edge of the figure.

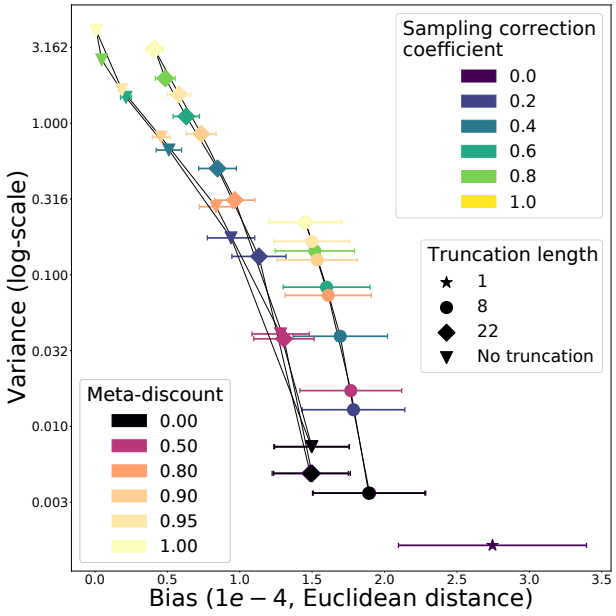


Figure B.5: The bias and variance of two different weighting schemes for the sampling correction terms are explored.

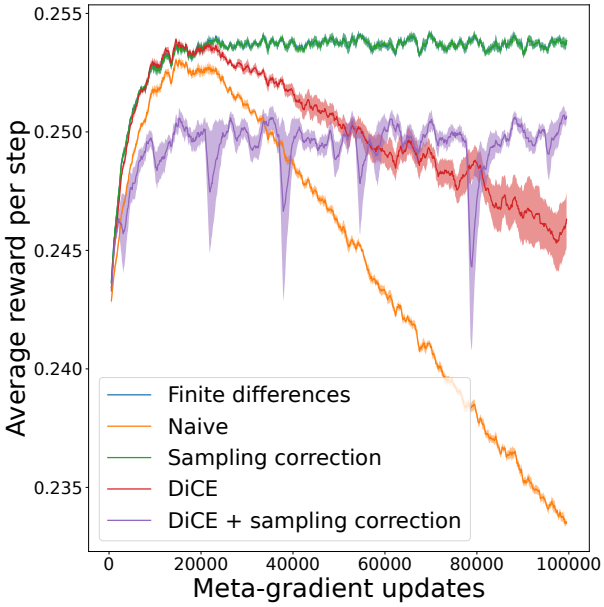


Figure B.6: Learning curves for the meta-gradient estimators in the bandit setting. The shading of the learning curves shows the standard error across random seeds. Note that the learning curves for the finite differences estimator and sampling corrected estimator are almost perfectly overlapped in this setting.

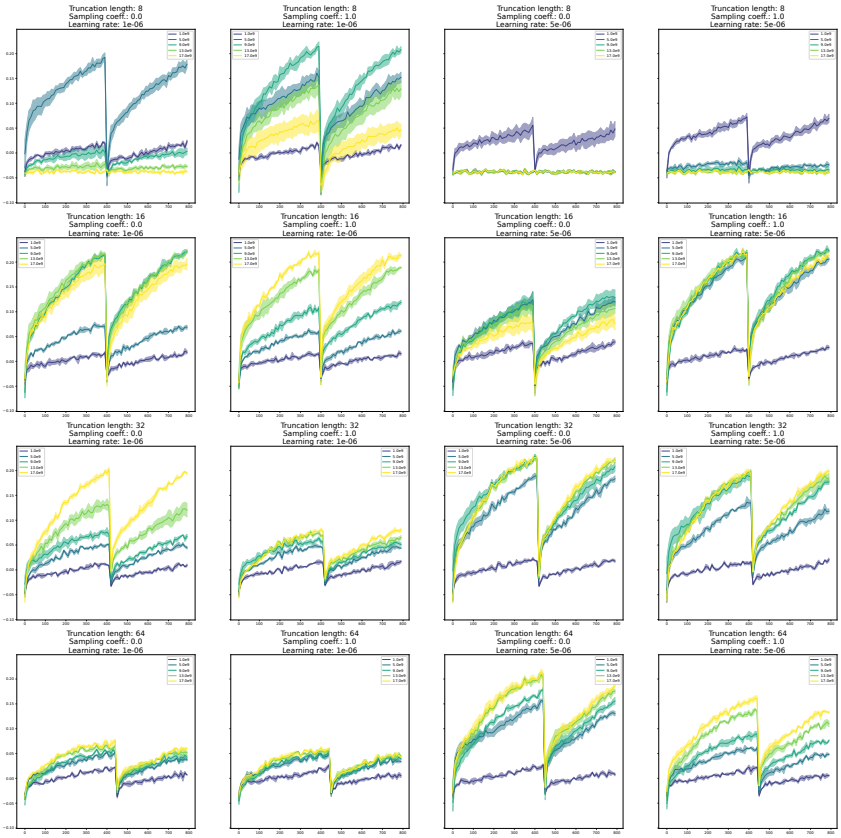


Figure B.7: Snapshots into the reward per timestep in the gridworld experiment. The figures are organized into columns by values of sampling correction coefficient and outer-loop learning rate and into rows by truncation lengths.

B.7 Additional Experimental Results

The meta-gradient estimators are compared in terms of cosine similarity in the left panel of figure B.4. The two weighting schemes for the sampling correction are compared in the right panel of figure B.5. Learning curves for the truncated meta-gradient estimators using SGD as the optimizer are shown in figure B.2. The truncated learning curves with ADAM as the optimizer are shown again in figure B.3, with the added curves for truncation length 1. The large error in the curves for the higher truncation lengths are due to some of the seeds becoming unstable during training and deviating far from the optimal region in a single gradient step. This effect can be reduced in practice by using adaptive optimizers such as Adam and gradient clipping. The learning curves with the different gradient estimators in the

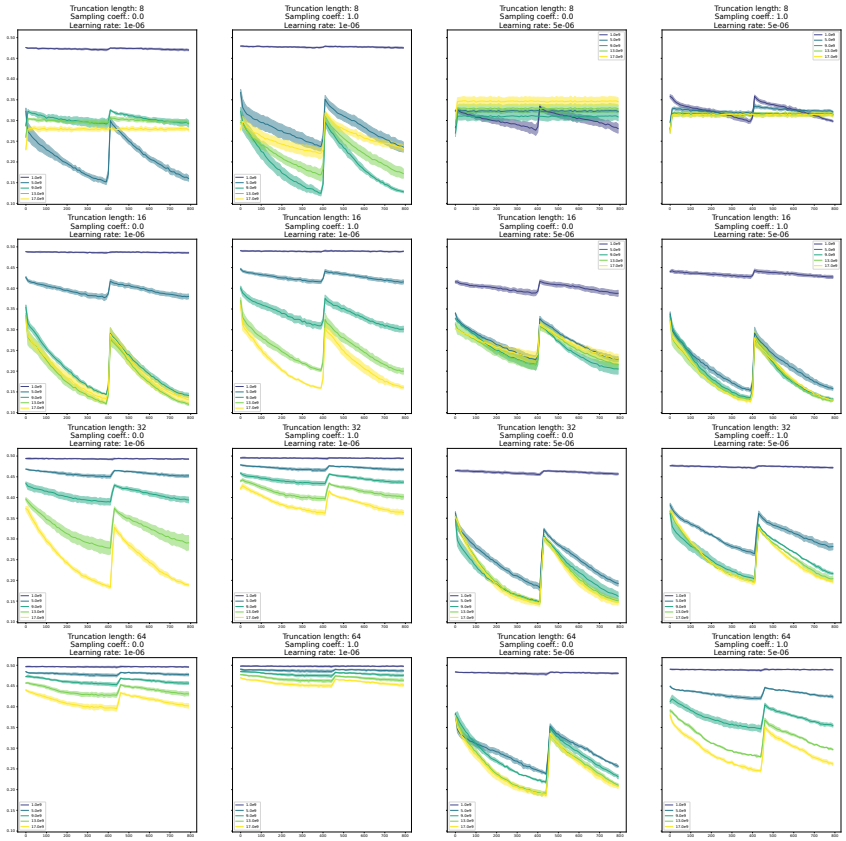


Figure B.8: Snapshots into the entropy regularizer coefficient per timestep in the gridworld experiment. The figures are organized into columns by values of sampling correction coefficient and outer-loop learning rate and into rows by truncation lengths.

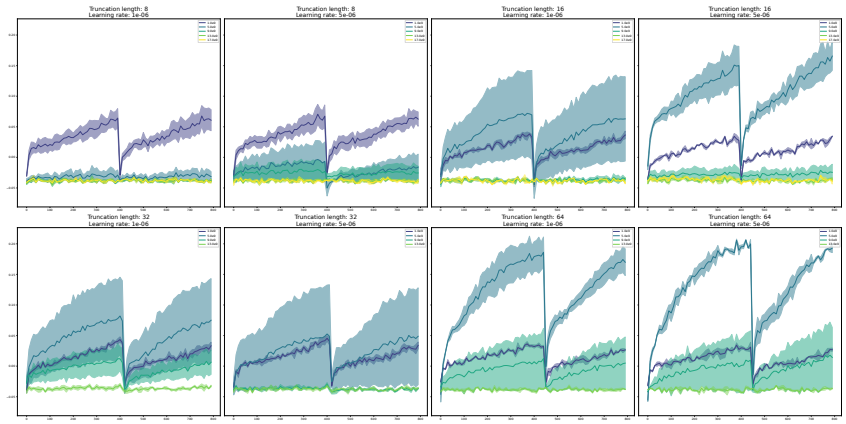


Figure B.9: Snapshots into the reward per timestep in the gridworld experiment with evolution strategies. The figures are organized into columns by values of sampling correction coefficient and outer-loop learning rate and into rows by truncation lengths.

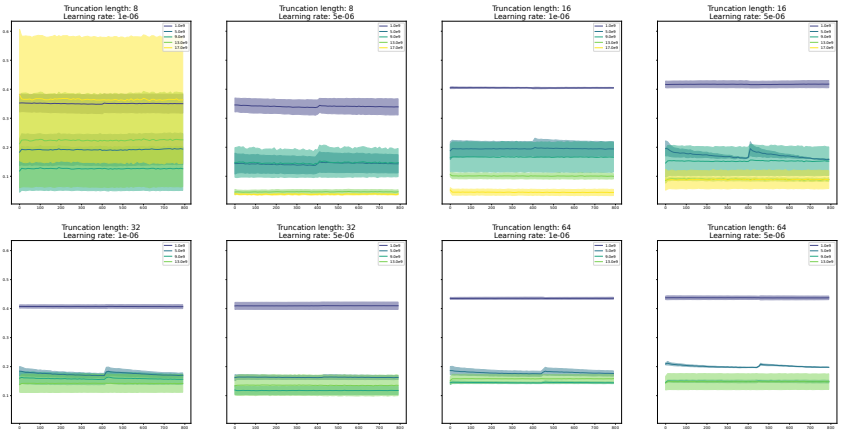


Figure B.10: Snapshots into the entropy regularizer coefficient per timestep in the gridworld experiment with evolution strategies. The figures are organized into columns by values of sampling correction coefficient and outer-loop learning rate and into rows by truncation lengths.

untruncated bandit setting are shown in figure B.6. In the meta-learning curves, all algorithms start from the same initialization. The gradients of the sampling corrected estimator are similar enough to the finite differences gradient in this setting that their respective learning curves are almost identical. Figures B.7 and B.8 show the reward per timestep and entropy coefficient respectively of the gridworld experiment in 5.4.4 evaluated at different snapshots during the meta-training. The figures show two reward flip intervals. Figure B.11 shows bias and variance for two initializations with a truncated estimator where the bias increases with the sampling correction coefficient for one initialization and decreases for the other.

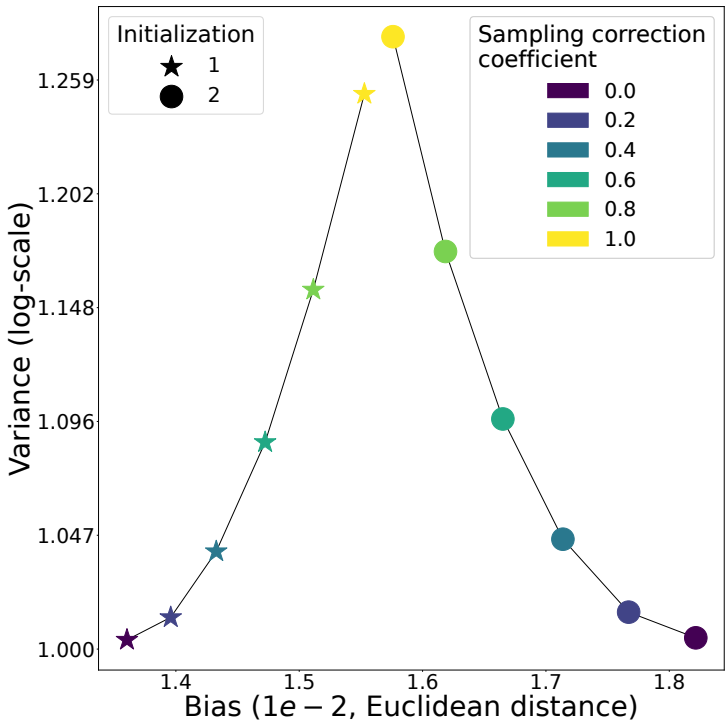


Figure B.11: Bias and variance for two initializations with a truncated estimator where the bias increases with the sampling correction coefficient for one initialization and decreases for the other

C

Deconfounding Imitation Learning with Variational Inference

Contents

C.1 Confounding and Stochasticity

The behavior success of the conditional and interventional policies depend on the stochasticity of the environment and the expert, as this affects the evidence for the inference of the latent variable. If most evidence comes from the dynamics, which would e. g. be the case when the expert policy is very noisy and the dynamics deterministic, we expect the conditional and the interventional policies to be similar. However, when most evidence comes from the policy decisions, we expect large delusion.

See figure figure C.1 for a plot of the rewards of the various policies given varying amounts of stochasticity. Instead of the fixed coefficients of equation (7.2), we

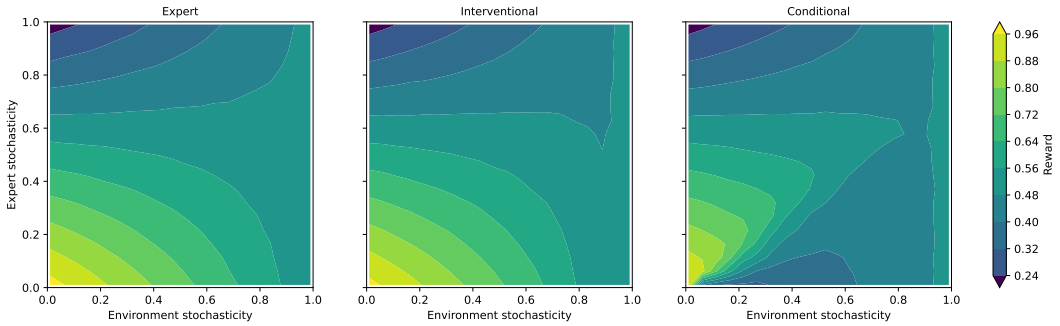


Figure C.1: Conditional and interventional policies in the bandit environment equation (7.2) as a function of environment stochasticity and expert stochasticity. We show the expected success probability for the expert (left), the interventional policy (middle), and the conditional policy (right). Along the axes, we vary the expert policy from deterministic (bottom) to stochastic (top) and the environment transitions from deterministic (left) to stochastic (right); see text for details. The interventional policy performs similar to the expert, while the conditional policy performs worse due to latent confounding when the environment stochasticity is larger than the expert stochasticity.

vary $\pi_{\text{exp}}(a_t = \theta)$ between 1 (deterministic expert) and $\frac{1}{5}$ (maximally stochastic expert) and $p(s_{t+1} = 1|a_t = \theta)$ between 1 (deterministic environment) and $\frac{1}{2}$ (maximally stochastic environment).

In the left panel, we show the expected success probability of the expert policy: as expected, it is highest when both the expert and the environment are deterministic. The interventional policy of equation (7.3) performs similarly. However, the conditional policy, shown in the right panel, is only able to match the expert behavior if the stochasticity in the expert policy outweighs the stochasticity in the environment. When the environment is stochastic and the expert comparably deterministic, the conditional policy (and thus naive BC) infers latents largely from past actions, suffers from delusions, and fails to imitate.

C.2 Training Deconfounded Imitators

Algorithm 3 shows how an agent is trained in practice from expert data in the presence of latent confounders. Algorithm 4 shows the test time implementation of the deconfounded imitation learners, which alternate between updating a posterior belief over the latent and acting under the current belief.

Algorithm 3 Training deconfounded imitators

Require: Initial parameters of the imitation policy η, inference model ϕ, and dynamics model ψ; expert dataset $\{\tau_e^j\}$, MDP family $\mathcal{M}_\theta$, true latent distribution $p(\theta)$, learning rates α_1, α_2, α_3.

repeat
- $\theta \sim p(\theta)$, $s_0 \sim p_0(s_0)$
- $\tau = s_0$, $t = 0$
- **for** $t \leq H$ **do**
  - $\hat{\theta}_t \sim q_\phi(\hat{\theta}_t \mid \tau_{:t})$ ▷ Infer the latent for trajectory
  - $a_t \sim \pi_\eta(a_t \mid s_t, \hat{\theta})$ ▷ Sample action from a Markov policy
  - $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t, \theta)$ ▷ True dynamics
  - Append (a_t, s_{t+1}) to τ
  - $t \leftarrow t + 1$
- **end for**
- $\phi \leftarrow \phi - \alpha_1 \nabla_\phi \hat{\mathcal{L}}_{\mathrm{VI}}(\tau)$ ▷ Sample estimate of equation (7.10)
- $\psi \leftarrow \psi - \alpha_2 \nabla_\psi \hat{\mathcal{L}}_{\mathrm{VI}}(\tau)$
- $\hat{\theta}_H^j \sim q_\phi(\hat{\theta}_H^j \mid \tau_e^j)$ ▷ Infer latent from j-th expert trajectory
- $\eta \leftarrow \eta - \alpha_3 \nabla_\eta \sum_j \sum_{s,a \in \tau_e^j} \log \pi_\eta(a \mid s, \hat{\theta}_H^j)$

until done

Algorithm 4 Deconfounded imitators at test time

Require: Parameters of the policy η and inference model ϕ; MDP $\mathcal{M}_\theta$ with ground-truth latent θ.

$s_0 \sim p_0(s_0)$
$\tau \leftarrow s_0$, $t \leftarrow 0$
for $t \leq H$ **do**
- $\hat{\theta}_t \sim q_\phi(\hat{\theta}_t \mid \tau_{:t})$ ▷ Infer the latent for trajectory
- $a_t \sim \pi_\eta(a_t \mid s_t, \hat{\theta}_t)$ ▷ Condition on inferred latent
- $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t, \theta)$ ▷ True dynamics
- Append (a_t, s_{t+1}) to τ
- $t \leftarrow t + 1$

end for

C.3 Deconfounded Imitation Learning from Expert Data Alone

As discussed in section 7.4, when suitable demonstrations from the expert are available, the interventional policy is identifiable from the expert data alone, without access to the environment or the expert. In this section, we present a practical algorithm for learning the deconfounded imitation policy from such an expert dataset. At test time, the agent works the same way as the Algorithm 3 presented in the paper.

Training an inference model directly on the expert demonstration faces the same problem as naive imitation learning, i. e., the trained model takes the expert's actions as evidence for the latent. However, by the assumption, that the expert is uniquely determined by the environment dynamics, we can directly learn the conditional policy and dynamics model explaining the expert trajectories from the demonstrations because a latent that explains the dynamics also explains the expert. To train such models, we use variational inference to learn a trajectory encoder $q_{\phi_{\text{off}}}$, which infers the latent for the expert trajectories, and a factorized decoder, which reconstructs the dynamics of the environment and the expert's policy using networks p_ψ and π_η respectively. The variational inference objective is given by

$$\mathcal{L}_{\text{off},i} = \mathbb{E}_{\hat{\theta}\sim q_\phi(\hat{\theta}|\tau_e^i)}\left[\sum_{t=0}^{H} \log p_\psi(s_{t+1} \mid s_t, a_t, \hat{\theta}) + \log \pi_\eta(a_t|s_t, \hat{\theta})\right] - \beta D_{KL}\Big(q_{\phi_{\text{off}}}(\hat{\theta} \mid \tau_e^i) \,\Big\|\, p(\hat{\theta})\Big), \tag{C.1}$$

which, unlike the objective in the main paper, represents a VAE where the encoder $q_{\phi_{\text{off}}}$ takes as input the full expert trajectory τ_e^i, and the decoder decodes both the action and transition probabilities throughout the trajectory.

This gives us a way for training the conditional policy imitating the experts in the demonstrations and a dynamics model. However, we cannot directly use the learned inference model $q_{\phi_{\text{off}}}$ for implementing the interventional policy, because it takes the expert's actions as evidence for the latent. The Algorithm 3 works by separately learning an inference model from interactions with the environment and using that inference model for deconfounding the expert trajectories. When we do not have sampling access to the environment, we cannot learn the inference model directly. Instead, we observe that one factor of the decoder used for training the inference model is a dynamics model of the environment conditional on the predicted latent. Therefore, we can use it to generate synthetic trajectories for training an online inference model $q_{\phi_{\text{on}}}$ to minimize the online variational inference objective given in the main paper. The online inference model can then be used for implementing the interventional policy similarly as in Algorithm 4. The full offline dynamics learning algorithm is presented in Algorithm 5.

Algorithm 5 Training deconfounded imitators, offline variant

Require: The initial parameters of the imitation policy η, offline inference model ϕ_{off}, online inference model ϕ_{on}, dynamics model ψ, prior distribution for the learned latent space $p(\tilde{\theta})$, a dataset of expert trajectories $\{\tau_e^i\}$, an MDP $(\mathcal{S}, \mathcal{A}, p, p_0, H)$, learning rates α_1, α_2, α_3, and α_4.

while not done **do**
- $\tilde{\theta} \sim p(\tilde{\theta})$ ▷ Sample from the prior
- $s_0 \sim p_0(s_0)$ ▷ Sample from a learned model or expert data
- $\tau_{\text{synth}} = s_0$, $t = 0$
- **for** $t \leq H$ **do**
  - $a_t \sim \pi(a_t \mid s_t)$ ▷ Sample action from a Markov policy
  - $s_{t+1} = p_\psi(s_{t+1} \mid s_t, a_t, \tilde{\theta})$ ▷ Dynamics model
  - Append (a_t, s_{t+1}) to τ_{synth}
  - $t = t + 1$
- **end for**
- $\phi_{\text{on}} = \phi_{\text{on}} - \alpha_1 \nabla_{\phi_{\text{on}}} \hat{\mathcal{L}}(\tau_{\text{synth}})$ ▷ Train the online model on equation (7.10).
- $\phi_{\text{off}} = \phi_{\text{off}} - \alpha_2 \nabla_{\phi_{\text{off}}} \sum_j \hat{\mathcal{L}}_{\text{off}}(\tau_e^j)$ ▷ Train the offline model on equation (C.1).
- $\psi = \psi - \alpha_3 \nabla_\psi \sum_j \hat{\mathcal{L}}_{\text{off}}(\tau_e^j)$ ▷ Train the dynamics model on equation (C.1).
- $\eta = \eta - \alpha_4 \nabla_\eta \sum_j \hat{\mathcal{L}}_{\text{off}}(\tau_e^j)$ ▷ Train the policy on equation (C.1).

end while

C.4 Multi-Armed Bandit Experiment

Implementation Details The inference model q_ϕ is implemented as an RNN with GRU architecture [Cho et al., 2014] with a hidden layer of 256 units. Before the RNN, the observation is preprocessed by an MLP with two hidden layers of size 256 units and output size 32. The action is preprocessed by a linear transformation to a 32 dimensional vector. The outputs of the RNN are processed by a linear transformation to a vector which parametrizes the latent distribution. The latent distribution is a 256 dimensional Gaussian. One half of the predicted vector represents the mean of the latent distribution and the other half, after softplus activation has been applied to it represents the variance.

The decoder is an MLP with two hidden layers of size 256 and a linear output layer. It uses the same input preprocessing networks for the observations and actions as the inference model. The policy is an MLP, which takes the latent sample, and an observation as inputs. It uses the same observation embedding network as the other networks and then has two hidden layers with 256 units each. The naive

Hyperparameter	Value
Episode length	100
Imitation training steps	5000
Dynamics model training batch size (full episodes)	100
Imitation training batch size (full episodes)	100
Behavioral cloning learning rate	0.001
Variational inference learning rate	0.0001
KL coefficient (β)	0.001

Table C.1: Hyperparameters for the deconfounded behavioral cloning and naive behavioral cloning algorithms

BC baseline uses the same network architecture as the deconfounded algorithm, except it does not represent the belief as a probabilistic latent variable and therefore there is no sampling step. It just directly passes output of the trajectory encoder as the input to the policy network. All of the MLPs use ReLU activations. All networks are optimized using the Adam optimizer [Kingma and Ba, 2014] with default settings from PyTorch [Paszke et al., 2019], except for the learning rate.

Hyperparameter Settings The hyperparameters used for the learning algorithms are presented in table C.1.

Computing the Ground Truth Policies All of the probabilities relevant to the bandit problem are known exactly from the definition of the problem and the conditional and interventional policies given in the main paper. Using these probabilities we can compute the true conditional and interventional policies, allowing us to compare the learned algorithms to the relevant optimal policies.

In practice, the true belief over theta can be computed for any trajectory as follows

$$\begin{aligned}
&\log p(\hat{\theta}_0) = \log \frac{1}{5}, \\
&\log p(\hat{\theta}_{t+1}[A_t]) = \begin{cases} \log p(\hat{\theta}_t[A_t]) + s_t \log \frac{3}{4} + (1 - s_t) \log \frac{1}{4} & \text{if } A_t = a_t \\ \log p(\hat{\theta}_t[A_t]) + s_t \log \frac{1}{4} + (1 - s_t) \log \frac{3}{4} & \text{otherwise} \end{cases}.
\end{aligned} \tag{C.2}$$

The true interventional policy can then be computed by sampling a belief $\hat{\theta}_t \sim \log p(\hat{\theta}_t)$, and sampling an action from $\pi_{\text{exp}}(a_t|s_t, \hat{\theta}_t)$. This can be seen as a Thompson

sampling policy [Thompson, 1933], which acts optimally given its current belief of the task. The true conditional policy is computed similarly, except taking the actions as evidence for the latent is added to the update

$$\log p(\hat{\theta}_{t+1}[A_t]) = \begin{cases} \log p(\hat{\theta}_t)[A_t] + s_t \log \frac{3}{4} + (1 - s_t) \log \frac{1}{4} + \log \frac{6}{10} & \text{if } A_t = a_t \\ \log p(\hat{\theta}_t)[A_t] + s_t \log \frac{1}{4} + (1 - s_t) \log \frac{3}{4} + \log \frac{1}{10} & \text{otherwise} \end{cases}. \tag{C.3}$$

C.5 Confounded MDPs

Environment Descriptions For `LunarLander-v2` [Brockman et al., 2016], we consider a modified version with unknown key bindings: a latent θ specifies a permutation in the map between two of the the agent's actions and the behaviors (firing the left and right engines of the space craft). This permutation is known to the expert, but not the imitator. In addition, we make the environment stochastic: at each step, with a probability of 0.15, a uniformly chosen random action is performed instead of the action chosen by the agent. The confounding problem arises because the expert's actions are temporally correlated, for example it may select the same action multiple times to adjust the lander's attitude. A naive recurrent BC policy may thus use past actions to predict future actions - causing confounding if the controls are swapped, as a wrong initial guess influences future decisions.

For `HalfCheetahBulletEnv-v0` [Coumans and Bai, 2016–2021], we modify the environment similarly to Swamy et al. [2022b] by varying the target speed. The expert observes the true target speed while the imitator only observes an indicator, which shows whether it is running faster or slower than the target speed. The expert smoothly tends to the target speed. This leads to correlated actions, in which the trend of past speeds can be extrapolated. A confounded naive BC imitator may attempt to extrapolate a random trend of its past speeds. Contrary to Swamy et al. [2022b], we find that the deterministic version of this environment does not necessarily differentiate between naive BC and DAgger. Therefore, we make the indicator stochastic by randomizing it 20% of the time. For a detailed discussion about the effect of stochasticity on `HalfCheetahBulletEnv-v0`, see below.

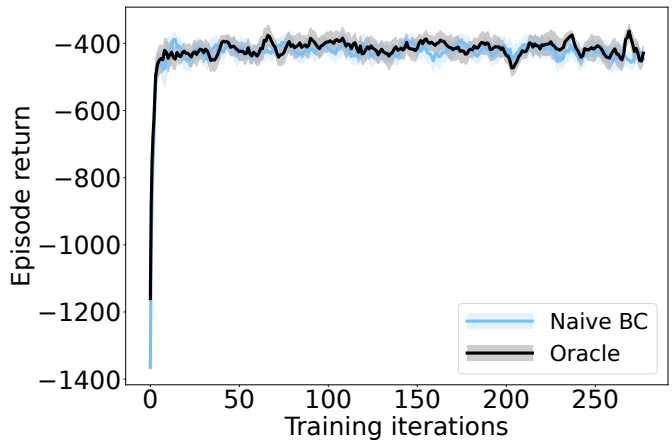


Figure C.2: Comparing naive BC and oracle in a deterministic variant of `HalfCheetahBulletEnv-v0`. Most of the difference in the episodic returns here compared to the returns reported in Swamy et al. [2022b] follows from their results adding a constant 1000 to the episodic return for plotting.

In `AntGoal-v0` [Todorov et al., 2012], we consider a version of the classic ant robot locomotion environment, where the task is to run to a goal randomly sampled from within a circle around the starting point. This is similar to the ant environment considered by Zintgraf et al. [2020]. We make the goal radius $\frac{10}{3}$ times larger to make the locomotion task more challenging and remove the control cost. The imitators receive noisy observations of the length and angle of the vector between the ant and the goal in the global coordinate frame. Gaussian noise with scale 0.1 is added to the indicator. The expert knows the true goal location and starts moving toward it immediately, making its actions correlated across time. The confounding arises because on the expert trajectories, the goal direction and distance can be inferred from the expert actions and states without considering the noisy indicator.

Deterministic vs Stochastic variant of HalfCheetah. We test the different imitation learning algorithms on the `HalfCheetahBulletEnv-v0` environment from Swamy et al. [2022b]. We found that running our implementation of naive BC matches the performance of our implementation of DAgger, which is inconsistent with the results of Swamy et al. [2022b], where they report that DAgger outperforms naive BC. See figure C.2 for learning curves. One reason for why their results show a difference between the algorithms may be that their implementation of BC

may suffer from covariate shift resulting from it having been trained on a small number of samples compared to DAgger. However, if that is the case, it is not an example of the confounding problem we are interested in. To make sure that we are testing for the ability of the algorithms to deal with the confounding problem in this environment, we make the indicator variable stochastic. The results for the stochatic variant are reported in the main paper.

Implementation Details For LunarLander, HalfCheetah, and Ant we use the same network architecture as the bandit experiments with the following changes. The output of the encoder parametrizes the mean and the log-variance of the latent distribution. There is no parameter sharing between the different input processing networks in the encoder, policy, and decoder. The RNN has a hidden size of 500. The MLPs have two hidden layers of size 128. The action embedding has size 128. The latent distribution is a 64 dimensional Gaussian for LunarLander and HalfCheetah, and 8 dimensional Gaussian for Ant. In practice, the latent dimensionality does not matter very much. The MLPs use ELU [Clevert et al., 2015] activations followed by LayerNorm [Ba et al., 2016]. The networks are optimized with AdamW [Loshchilov and Hutter, 2017].

The experts are trained using a PPO implementation by Raffin et al. [2021] with hyperparameters from Raffin [2020]. Additionally, a running normalization layer [Raffin et al., 2021] is applied to the observations and fixed after expert training to make the distribution of the observations easier to learn the decoder on.

On each iteration of the training algorithm, a batch of new episodes are collected with the appropriate policy or policies. Naive BC only collects data using the expert policy. DAgger collects data using a policy defined by a mixture of the expert and imitator policies, which is linearly annealed from the expert policy to the imitator policy during the training. Deconfounded algorithm collects a batch episodes with the expert policy for BC and a batch episodes with the imitation policy for training the encoder. These trajectories are saved in circular buffers that

Hyperparameter	Value
Top-level training iterations	300
Imitation / inference training steps per top-level iteration	100
Dynamics model training batch size (full episodes)	16
Imitation training batch size (full episodes)	16
BC learning rate	0.0003
Variational inference learning rate	0.0003
KL coefficient (β)	1.0

Table C.2: Hyperparameters for the deconfounded BC, DAgger, and naive BC algorithms for `LunarLander-v2`, `HalfCheetahBulletEnv-v0`, and `AntGoal-v0` environments. We fix the episode length for `LunarLander-v2` to 500 steps, for `HalfCheetahBulletEnv-v0` to 1000 steps, and for `AntGoal-v0` to 200 steps.

hold $2 * 10^6$ transitions. After each data collection step, the imitator (and encoder) are updated for 100 steps with the update function corresponding to the algorithm.

We implemented GAIL closely following a popular publicly available implementation and using recurrent PPO from `stable-baseline3` [Raffin et al., 2021] as the RL algorithm. We use the same data sampling pipeline as the deconfounded imitation learning for GAIL. Deconfounded algorithm collects a batch episodes with the expert policy for BC and a batch episodes with the imitation policy for training the encoder and stores those samples in separate buffers. The network architectures are the same as for the other policies, where applicable. We found that getting GAIL to work at all in our environments, we had to increase the number of episodes sampled between each update from 16 to 64. Furthermore, we found that GAIL did not work with reward function parameterized as a function of (s, a, s') in our environments, but worked better as a function of just (s').

Hyperparameter Settings The hyperparameters used for the learning algorithms are presented in table C.2.

Hyperparameter	Value
Top-level training iterations	300
PPO training iterations per algorithm iteration	1
PPO batch size (timesteps)	400
PPO gradient max norm	0.5
PPO normalize advantage	True
PPO epochs per training iterations	10
PPO discount rate γ	0.99
PPO GAE-λ	0.95
PPO learning rate	0.0003
PPO episodes per training iteration	64
Discriminator training steps per top-level iteration	20
Discriminator batch size (full episodes)	16
Discriminator maximum gradient norm	100

Table C.3: Hyperparameter settings for GAIL.

D

A Bayesian Solution To The Imitation Gap

Contents

Supplementary Material

D.1 Summary of Distributions in BIG

Due to the diverse nature of the sources of input information, inferring the reward posterior requires the agent to learn and maintain several distributions over random variables, we summarize them in table D.1.

Table D.1: Summary of the distributions involved in BIG. This table supplements the diagram in figure 8.2.

DISTRIBUTION	NAME	DESCRIPTION	SAMPLES TO LEARN
$p(\omega)$	Reward Parameter Prior	Incorporates prior knowledge in reward parameterization	None
$p(k)$	Cost of Exploration Prior	Incorporates prior knowledge in rewards $r(s,a) = kr_{\max}$ over exploratory state-actions	None
$p(\theta)$	Contextual Prior	Characterizes uncertainty in θ a priori at test time	$\mathcal{D}_{\text{Sim}}$ - samples from the CMDP simulator
$p(a\|s,\theta,\omega)$	Model Expert Policy	Model of optimal policy for expert in CMDP $\mathcal{M}(\theta)$ with reward parameters ω	Contextual successor features learned from simulator and $\mathcal{D}_{\text{Expert}}$
$p(\omega\|\mathcal{D}_{\text{Expert}})$	Expert reward Posterior	Updates the reward parameter prior using expert data	$\mathcal{D}_{\text{Expert}}$ dataset of expert trajectories
$p(\theta_i\|\tau_i)$	Contextual Posterior	Characterizes uncertainty over which latent variable θ_i agent i was assigned	τ_i - each expert or exploratory agent's trajectory
$p^{\text{IRL+COE}}_{\text{Bayes}}(r\|s,a)$	Bayesian Reward Distribution	Incorporates epistemic uncertainty from $p(\omega\|\mathcal{D}_{\text{Expert}})$ and $p(k)$ into reward model	None

D.2 Bayesian Successor Feature Learning

Consider the optimal Q-function $Q^\star(s,a,\theta,\omega)$, which satisfies the optimal contextual Bellman equation: $\mathcal{B}^\star[Q^\star](s,a,\theta,\omega) = Q^\star(s,a,\theta,\omega)$ where:

$$\mathcal{B}^\star[Q^\star](s,a,\theta,\omega) \coloneqq \nu(s,a)^\top \omega \tag{D.1}$$

$$+ \gamma \mathbb{E}_{s' \sim p(s'|s,a,\theta)} \left[\sup_{a'} Q^\star(s',a',\theta,\omega) \right]. \tag{D.2}$$

As the expert selects actions $a \in \arg\max_{a'} Q^\star(s,a',\theta,\omega)$, learning $Q^\star(s,a,\theta,\omega)$ is sufficient for modeling the set of expert policies, from which the true reward parametrization can be inferred.

Learning with Expert Data. Consider the Bellman equation under the expert policy $\pi^\star(a'|s',\theta_i)$. The successor feature representation Ψ_ϕ should satisfy:

$$\Psi_\phi(s,a,\theta_i)^\top \omega^\star = \nu(s,a)^\top \omega^\star + \gamma \mathbb{E}_{s' \sim p(s'|s,a,\theta_i), a' \sim \pi^\star(a'|s',\theta_i)} \left[\Psi_\phi(s',a',\theta_i) \right]^\top \omega^\star. \tag{D.3}$$

We can factor $\omega^\star$ from the Bellman equation and then solve:

$$\Psi_\phi(s,a,\theta_i) = \nu(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a,\theta_i), a' \sim \pi^\star(a'|s',\theta_i)} \left[\Psi_\phi(s',a',\theta_i) \right]. \tag{D.4}$$

This yields the objective for each expert trajectory τ_i:

$$\mathcal{L}_{\text{Expert}}(\phi; \tau_i) \coloneqq \mathbb{E}_{s,a\sim\text{Unif}(\tau_i),\theta_i\sim p(\theta_i|\tau_i)}\Big[\|\Psi_\phi(s,a,\theta_i) - (\nu(s,a) \tag{D.5}$$

$$+ \gamma\mathbb{E}_{s'\sim p(s'|s,a,\theta_i),a'\sim\pi^\star(a'|s',\theta_i)}\left[\Psi_\phi(s',a',\theta_i)\right])\|\Big] \,. \tag{D.6}$$

where $\text{Unif}(\tau_i)$ is a uniform distribution over the state-action pairs in trajectory τ_i. Minimizing $\mathcal{L}_{\text{Expert}}(\phi; \tau_i)$ for each trajectory ensures that the successor feature representation is consistent with the expert demonstrations. Minimizing $\mathcal{L}_{\text{Expert}}(\phi; \tau_i)$ can be carried out using a semi-gradient approach, thereby avoiding the need for two samples from the expert policy and state-transition distribution:

$$\phi \leftarrow \phi + \eta_\phi \nabla_\phi \Psi_\phi(s,a,\theta_i)(\nu(s,a) + \gamma\Psi_\phi(s',a',\theta_i) - \Psi_\phi(s,a,\theta_i)). \tag{D.7}$$

As the latent contextual variable θ_i is never observed, we must infer a posterior over its value $p(\theta_i|\tau_i)$. It may seem that this could be avoided via IRL using the prior-averaged transitions $\mathbb{E}_{\theta\sim p(\theta)}\left[p(s'|s,a,\phi)\right]$. However, we provide a simple counterexample in appendix D.3 demonstrating that a prior-averaged approach does not account for the true reward ordering in the underlying space of CMDPs. Sampling $\theta_i \sim p(\theta_i|\tau_i)$ from the posterior over the expert's contextual variable is typically intractable, so we use variational inference instead, as detailed in appendix D.5.

Learning with a Simulator. In addition to expert demonstrations, we also have access to samples from the CMDP simulator. This allows us to learn Ψ_ϕ over state-action pairs where the expert has not provided demonstrations. We sample a dataset $\mathcal{D}_{\text{Simulator}} \coloneqq \{\tau_i\}_{i=1}^{N_{\text{Simulator}}}$ of $N_{\text{Simulator}}$ trajectories from the simulator: the simulator samples an MDP from the prior, and then an exploration policy π_{Explore} interacts with the corresponding MDP, observing state-action-state transitions. In chapter 8, we use an ϵ-greedy exploration policy where the greedy actions $a' \in \arg\max_{a'} \Psi_\phi(s',a',\theta_i)^\top\omega$ are taken with probability $1-\epsilon$ then uniformly otherwise. More sophisticated approaches such as a policy that maximizes the entropy of the ergodic, discounted state-action occupancy distribution [Hazan et al., 2019] would also be appropriate, especially in larger domains. However, our experiments demonstrate the ϵ-greedy policy suffices for learning successor features in our setting.

Once samples have been obtained, we infer the posterior $p(\theta_i|\tau_i)$ over the MDP that the exploratory agent was assigned for each MDP $i \in [1 : N_{\text{Simulator}}]$. Like with the expert data, our goal is to ensure that the successor feature representation Ψ_ϕ satisfies a Bellman equation. In the target, we choose the next action that maximizes the Q-function for a given ω, $a' \in \arg\max_{a'} \Psi_\phi(s', a', \theta_i)^\top \omega$. This yields the objective for each MDP i:

$$\mathcal{L}_{\text{Simulator}}(\phi; \tau_i, \omega) \coloneqq \mathbb{E}_{s,a\sim\text{Unif}(\tau_i),\theta_i\sim p(\theta_i|\tau_i)}\Big[\tag{D.8}$$

$$\Big(\Psi_\phi(s, a, \theta_i) - (\nu(s, a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a,\theta_i)} \left[\Psi_\phi(s', a', \theta_i)\right]\Big)^2\Big]. \tag{D.9}$$

where $\text{Unif}(\tau_i)$ is a uniform distribution over the state-action pairs in trajectory τ_i. We optimize this objective using the following TD update:

$$\phi \leftarrow \phi + \eta_\phi \nabla_\phi \Psi_\phi(s, a, \theta_i) \cdot (\nu(s, a) + \gamma \Psi_{\phi'}(s', a', \theta_i) - \Psi_\phi(s, a, \theta_i)). \tag{D.10}$$

There are two differences between the updates for the exploratory data and the expert data. First, the exploratory data updates are off-policy. Due to the deadly triad [Sutton and Barto, 2018], using semi-gradients is not guaranteed to converge [Fellows et al., 2023b, Tsitsiklis and Van Roy, 1997]. We introduce a separate target network ϕ' that is updated periodically to stabilize the updates. Second, the updates depend on ω as the supremum acts over the dot product between the successor representation and the reward parametrization. As we require successor features to infer ω, we interleave learning both ω and ϕ in a nested optimization, using both expert and exploratory data with an initial burn-in period using only the expert data.

We combine both objectives into the single objective presented in section 8.3.1:

$$\mathcal{L}(\phi; \omega) = \mathcal{L}_{\text{Expert}}(\phi; \tau_i) + \beta \mathcal{L}_{\text{Simulator}}(\phi; \tau_i, \omega), \tag{D.11}$$

for some constant β. However, we note that the reward learning and successor feature learning depend on each other through the objective $\mathcal{L}_{\text{Simulator}}$ and reward update given by theorem 2, yielding a two-timescale learning problem. To enable convergence to a stable local optimum, the learning rates η_ϕ and η_ω are set such that $\eta_\omega < \eta_\phi$.

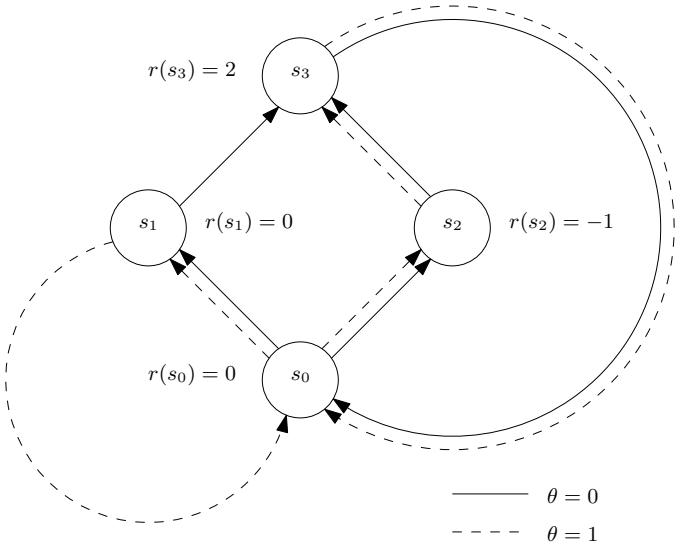


Figure D.1: Counterexample CMDP

D.3 The Need for Inference Over Context Variables

We consider a simple counterexample illustrated in figure D.1 with four states s_i and two possible hidden contexts denoted by θ. As shown in the figure, θ controls the environment transition. The simplest approach to IRL using the average MDP would map the experts' state visitation frequency η to reward value. This would assign rewards: $r(s_1) = c\eta$, $r(s_2) = c(1-\eta)$, $r(s_3) = c$, where c is an arbitrary finite positive constant. This means that the values of $r(s_1)$ and $r(s_2)$ are only determined by η and will give incorrect reward ordering $r(s_2) \geq r(s_1)$ for any $\eta \leq 0.5$. On the other hand, if a Bayesian approach is taken, it is possible to infer which MDP the agent was in. If the belief of a trajectory is weighted towards the expert being in $\theta = 0$, the expert's preference for $a_0 =$ left over $a_1 =$ right must be a consequence of the reward ordering $r(s_1) > r(s_2)$. Likewise, if the belief of a trajectory is weighted towards the expert being in $\theta = 1$, the expert's preference for $a_0 =$ right must imply that $r(s_2) + \gamma r(s_3) > r(s_1) + \gamma r(s_0)$. As $r(s_1) > r(s_2)$ will be inferred from the trajectories of experts in $\theta = 0$, this implies that $r(s_0) < r(s_3)$, and so a correct reward ordering will be learned regardless of η. By being Bayesian, these possible inferences condition on which θ_i the expert was in, and are explained

by $Q_{\zeta^\star}(s, a, \theta_i, \omega)$. We can best match ω so that is consistent with these inferences under the belief $p(\theta_i|\tau_i, \omega)$ in the expert's MDP. We thus conclude that compared to a fully Bayesian approach, naively using imitation learning on the prior-averaged MDP will not yield policies that account for reward ordering of the underlying MDP.

D.4 Bayesian Solution to the Imitation Gap

Finally, we present pseudocode for the Bayesian solution to the Imitation Gap (BIG) in algorithm 6. It takes as inputs the prior distributions and the expert dataset and produces a reward function $r_{\text{Bayes}}^{\text{IRL + COE}}$. In the main loop, data is collected from the simulated environment using an ϵ-greedy policy. We use ϵ-greedy for convenience as we found it to be an easy way to achieve suitable coverage of the state-action space in the experiments. The data collected using the random policy is used for learning the successor features in an off-policy RL algorithm. With an off-policy algorithm, assuming coverage of the whole state-action space, the exact kind of randomness does not matter for the kind of successor features we learn. The collected data together with the expert demonstrations are used to update the successor feature representation. The learned successor features are used for updating the reward parameters to maximize the likelihood of the expert data. After K steps of the main loop, the final reward function is computed by applying the cost-of-exploration refinement defined in section 8.3.3.

D.5 Approximate Inference

Whilst the full Bayesian approach outlined in section 8.3 is clearly desirable, there are several sources of intractability that prevent us from computing the Bayes-optimal policy $\pi^\star_{\text{Bayes}}$ exactly. Firstly, maintaining and inferring the posterior distributions is likely to be intractable for all but the simplest choice of likelihoods, which have insufficient capacity for representing the set of MDPs beyond contrived toy tasks. Secondly, marginalization using the posteriors likely will involve high dimensional integrals, which are computationally inefficient. Finally, solving the

Algorithm 6 Bayesian Solution to the Imitation Gap

Require: priors $p(\theta)$, $p(\omega)$, and $p_{\text{COE}}(r|\sigma, s, a)$, contextual posterior $p(\theta|\tau)$, dataset $\mathcal{D}_{\text{Expert}}$, reward scales $r_{\min}$ and $r_{\max}$, learning rates η_ω and η_ϕ, loss coefficient β, and number of steps K.
Initialize parameters ϕ for the successor features and ω for the reward.
Initialize an empty replay buffer $\mathcal{D}_{\text{Replay}}$.
for K steps **do**
  Sample a new MDP from simulator $\theta_i \sim \rho(\theta)$.
  Sample a trajectory τ_i in the MDP defined by θ_i using an epsilon-greedy policy w.r.t. the Q-function defined by $\Psi_\phi(s, a, \tilde{\theta}_i)^\top \omega$ for $\tilde{\theta}_i \sim p(\theta_i|\tau_i)$.
  Add the trajectory τ_i to the replay buffer $\mathcal{D}_{\text{Replay}}$.
  Update ϕ to minimise $\mathcal{L}(\phi; \omega)$ using $\mathcal{D}_{\text{Expert}}$ and $\mathcal{D}_{\text{Replay}}$.
  Update ω using equation (8.3) with $\mathcal{D}_{\text{Expert}}$.
end for
Rescale ω s.t. rewards lie in the range $[r_{\min}, r_{\max}]$.
Compute the predictive reward $r^{\text{IRL + COE}}_{\text{Bayes}}$. **return** $r^{\text{IRL + COE}}_{\text{Bayes}}$

planning problem in the BAMDP for every possible history to obtain a Bayes-optimal policy is notoriously challenging, even for very simple domains [Zintgraf et al., 2020]. We thus derive an algorithm that follows the methodology of the formal Bayesian approach outlined in section 8.3 whilst making necessary approximations from the powerful toolkit of variational inference to ensure tractability.

D.5.1 Tractable Prior and Likelihood Learning

Instead of attempting to infer the posterior $p(\phi|\mathcal{D}_{\text{simulator}})$ exactly, which may be intractable, we use a MAP approach instead to learn a point estimate $\phi^\star_{\text{MAP}}$. The justification for this is that we have access to a simulator from which a large number of samples can be drawn. The Bernstein-von Mises theorem specifies that in the limit of large data $K \to \infty$, $p(\phi|\mathcal{D}_{\text{simulator}}) \to \delta_{\phi^\star_{\text{MLE}}}(\phi)$ where $\phi^\star_{\text{MLE}}$ is the maximum likelihood estimator, so we expect $p(\phi|\mathcal{D}_{\text{simulator}}) \approx \delta_{\phi^\star_{\text{MLE}}}(\phi)$. In the same limit, $\phi^\star_{\text{MAP}} \to \phi^\star_{\text{MLE}}$. Our choice of using a MAP estimator over the MLE estimator is purely practical in that the prior can add regularization to stabilize learning, as is seen by the contribution to the MAP objective:

$$\mathcal{L}_{\text{MAP}}(\phi) = \sum_{i=1}^{K} \log p(\tau_i|\phi) + \log(\phi). \tag{D.12}$$

To find a tractable objective to maximize the MAP estimate, we introduce a variational distribution $q_{\chi_i}(\theta_i)$ parametrized by $\chi_i \in X$, for which:

$$\log p(\tau_i|\phi) = \int \log p(\tau_i|\phi) q_{\chi_i}(\theta_i) d\theta_i, \tag{D.13}$$

$$= \int \log \left(\frac{p(\tau_i, \theta_i|\phi)}{p(\theta_i|\tau_i, \phi)} \right) q_{\chi_i}(\theta_i) d\theta_i, \tag{D.14}$$

$$= \int \log \left(\frac{p(\tau_i, \theta_i|\phi)}{p(\theta_i|\tau_i, \phi)} \cdot \frac{q_{\chi_i}(\theta_i)}{q_{\chi_i}(\theta_i)} \right) q_{\chi_i}(\theta_i) d\theta_i, \tag{D.15}$$

$$= \int \left(\log p(\tau_i, \theta_i|\phi) - \log q_{\chi_i}(\theta_i) \right) q_{\chi_i}(\theta_i) d\theta_i - \int \frac{\log p(\theta_i|\tau_i, \phi)}{\log q_{\chi_i}(\theta_i)} q_{\chi_i}(\theta_i) d\theta_i, \tag{D.16}$$

$$= \mathcal{L}_{\text{ELBO}}(\phi, \chi_i) + \text{KL} q_{\chi_i} p(\cdot|\tau_i, \phi), \tag{D.17}$$

$$\implies \mathcal{L}_{\text{ELBO}}(\phi, \chi_i) = \log p(\tau_i|\phi) - \text{KL} q_{\chi_i} p(\cdot|\tau_i, \phi), \tag{D.18}$$

where $\mathcal{L}_{\text{ELBO}}(\cdot)$ denotes an evidence lower bound (ELBO). Now, assuming there exists some $\chi_i{}^\star \in X$ such that $q_{\chi_i{}^\star}(\theta_i) = p(\theta_i|\tau_i, \phi)$, then $\sup_{\chi_i \in X} \text{KL} q_{\chi_i} p(\cdot|\tau_i, \phi) = 0$, hence it follows from equation (D.18) that

$$\log p(\tau_i|\phi) = \sup_{\chi_i \in X} \mathcal{L}_{\text{ELBO}}(\phi, \chi_i). \tag{D.19}$$

Substituting into the MAP objective from equation (D.12) yields:

$$\mathcal{L}_{\text{MAP}}(\phi) = \sum_{i=1}^{K} \sup_{\chi_i \in X} \mathcal{L}_{\text{ELBO}}(\phi, \chi_i) + \log(\phi). \tag{D.20}$$

To find a $\phi^\star_{\text{MAP}} \in \arg\sup_{\phi \in \Phi} \mathcal{L}_{\text{MAP}}(\phi)$, we recognize that for each MDP sampled from the simulator indexed by $i \in [1 : K]$, we can minimize the following index-specific ELBO:

$$\mathcal{L}_{\text{ELBO}}(\phi, \chi_i) = \mathbb{E}_{\theta_i \sim q_{\chi_i}(\theta_i)} \left[\sum_{t=1}^{H_i} \log p(s_t|s_{t-1}, a_{t-1}, \theta_i) + \log p(\theta_i|\phi) - \log(q_{\chi_i}(\theta_i)) \right], \tag{D.21}$$

with respect to ϕ and χ_i. Here, Variational Auto-encoders [Kingma and Welling, 2014] (VAEs) offer a powerful framework for solving this problem via a variational expectation-maximization (EM) algorithm. In VAE parlance, the distribution $p(\tau_i|\theta_i, \phi)$ is known as the *decoder*, parametrized by $\phi \in \Phi$.

Algorithm 7 LearnPrior+Likelihood

Initialize ζ, χ, ϕ
for K steps **do**
  Sample new MDP from simulator $\theta \sim \rho(\theta)$
  Sample initial state $s_0 \sim p_0(s_0)$
  for $t \in [1 : H_i]$ **do**
    Sample action $a_{t-1} \sim d(a_{t-1})$
    Sample transition $s_t \sim p(s_t|s_{t-1}, a_{t-1}, \theta)$
    Sample gradient $g_\chi \sim \nabla_\psi \mathcal{L}_{\text{ELBO}}(\phi, \chi)$
    $\chi \leftarrow \chi + \alpha_\chi^t g_\chi$
    Sample gradient
    $g_\phi \sim \nabla_\phi \left(\mathcal{L}_{\text{ELBO}}(\phi, \chi) + \frac{1}{KH_i} \log p(\phi) \right)$
    $\phi \leftarrow \phi + \alpha_\phi^t g_\phi$
    $g_\zeta \sim \nabla_\zeta \text{MSBE}(\zeta)$
    $\zeta \leftarrow \zeta - \alpha_\zeta^t g_\zeta$
  end for
end forreturn ζ, ϕ

For each MDP θ_i, we train an *encoder* $q_{\chi_i}(\theta_i)$, parametrized by $\chi_i \in X$, that acts as a variational approximation to the posterior: $p(\theta_i|\tau_i, \phi)$.

To learn the prior parameters, we propose algorithm 7. Instead of specifying an encoder for each $q_{\chi_i}(\theta_i)$, we train a single encoder $q_\chi(\theta_i)$ online using an entire trajectory of samples from a specific MDP θ_i. Once training has finished for that MDP, we sample another θ_{i+1}, using χ as the initialization parameters for the new encoder, $q_\chi(\theta_{i+1})$. Moreover, as both learning the reward prior and minimizing the MSBE to learn the likelihood require samples from a simulator, we interleave both optimization problems using the same interactions with the simulator.

D.6 Proofs and Derivations

Lemma 1. *The gradient of the log normalization constant* $\log z(s, \omega, \theta_i)$ *is*

$$\nabla_\omega \log z(s, \omega, \theta_i) = \mathbb{E}_{a \sim p(a|s,\omega,\theta_i)} \left[\Psi^\star(s, a, \theta_i) \right]. \tag{D.22}$$

Proof. We assume that $\mathcal{A}$ is continuous. If $\mathcal{A}$ is discrete, we replace the Lebesgue measure λ with the counting measure, and our proof remains unchanged. Taking

derivatives directly yields our desired result:

$$\nabla_\omega \log z(s,\omega,\theta_i) = \nabla_\omega \log \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha}\Psi^\star(s,a,\theta_i)^\top \omega\right) d\lambda(a), \tag{D.23}$$

$$= \nabla_\omega \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha}\Psi^\star(s,a,\theta_i)^\top \omega\right) d\lambda(a) \cdot \frac{1}{\int_{\mathcal{A}} \exp\left(\frac{1}{\alpha}\Psi^\star(s,a,\theta_i)^\top \omega\right) d\lambda(a)}, \tag{D.24}$$

$$= \int_{\mathcal{A}} \Psi^\star(s,a,\theta_i) \frac{\exp\left(\frac{1}{\alpha}\Psi^\star(s,a,\theta_i)^\top \omega\right)}{\int_{\mathcal{A}} \exp\left(\frac{1}{\alpha}\Psi^\star(s,a,\theta_i)^\top \omega\right) d\lambda(a)} d\lambda(a), \tag{D.25}$$

$$= \int_{\mathcal{A}} \Psi^\star(s,a,\theta_i) p(a|s,\omega,\theta_i) d\lambda(a), \tag{D.26}$$

$$= \mathbb{E}_{a\sim p(a|s,\omega,\theta_i)} \left[\Psi^\star(s,a,\theta_i)\right]. \tag{D.27}$$

□

Theorem 1. *Define* $\varsigma_0^2 \coloneqq \frac{\sigma_0^2}{\alpha}$. *Using the Laplace approximation for the posterior, the approximate Bayesian reward parametrization* $\omega^\star_{Bayes} \approx \omega^\star_{Laplace}$ *can be found by carrying out the following stochastic gradient descent updates on the log-posterior:*

$$\omega \leftarrow \omega + \eta_\omega \left(N_{Expert} H_i \left(\Psi^\star(s_j,a_j,\theta_i) - \mathbb{E}_{a_i\sim p(a_i|s_j,\omega,\theta_i)} \left[\Psi^\star(s_j,a_i,\theta_i)\right] \right) - \frac{(\omega-\omega_0)}{\varsigma_0^2} \right). \tag{D.28}$$

Proof. Using Laplace's approximation, we fit a Gaussian to the posterior distribution with mean $\omega^\star_{\text{Laplace}} \in \arg\sup_{\omega\in\Omega} p(\omega|\mathcal{D}_{\text{Expert}})$. Equivalently, we can maximize the log posterior instead. Defining $g(\omega;\mathcal{D}_{\text{Expert}}) \coloneqq \nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}})$, from the definition of the gradient of a log:

$$g(\omega;\mathcal{D}_{\text{Expert}}) = \nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}}) = \nabla_\omega p(\omega|\mathcal{D}_{\text{Expert}}) \cdot \frac{1}{p(\omega|\mathcal{D}_{\text{Expert}})}. \tag{D.29}$$

We define the joint set of expert contextual variables as $\Theta_{\text{Expert}} \coloneqq \{\theta_1,\theta_2,...\theta_{N_{\text{Expert}}}\} \in \Theta^{N_{\text{Expert}}}$. Taking gradients of the posterior yields:

$$\nabla_\omega p(\omega|\mathcal{D}_{\text{Expert}}) = \int_{\Theta^{N_{\text{Expert}}}} \nabla_\omega p(\omega|\mathcal{D}_{\text{Expert}},\Theta_{\text{Expert}}) dP(\Theta_{\text{Expert}}|\mathcal{D}_{\text{Expert}}), \tag{D.30}$$

$$= \int_{\Theta^{N_{\text{Expert}}}} \frac{\nabla_\omega p(\omega|\mathcal{D}_{\text{Expert}},\Theta_{\text{Expert}})}{p(\omega|\mathcal{D}_{\text{Expert}},\Theta_{\text{Expert}})} p(\omega|\mathcal{D}_{\text{Expert}},\Theta_{\text{Expert}}) dP(\Theta_{\text{Expert}}|\mathcal{D}_{\text{Expert}}), \tag{D.31}$$

$$= \int_{\Theta^{N_{\text{Expert}}}} \nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}},\Theta_{\text{Expert}}) dP(\Theta_{\text{Expert}},\omega|\mathcal{D}_{\text{Expert}}), \tag{D.32}$$

$$= \int_{\Theta^{N_{\text{Expert}}}} \nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}},\Theta_{\text{Expert}}) dP(\Theta_{\text{Expert}}|\omega,\mathcal{D}_{\text{Expert}}) p(\omega|\mathcal{D}_{\text{Expert}}). \tag{D.33}$$

Substituting yields our desired result:

$$\nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}}) = \int_{\Theta^{N_{\text{Expert}}}} \nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}}, \Theta_{\text{Expert}}) dP(\Theta_{\text{Expert}}|\omega, \mathcal{D}_{\text{Expert}}). \tag{D.34}$$

Now,

$$p(\omega|\mathcal{D}_{\text{Expert}}, \Theta_{\text{Expert}}) = \frac{p(\mathcal{D}_{\text{Expert}}|\omega, \Theta_{\text{Expert}})p(\omega)}{\int p(\mathcal{D}_{\text{Expert}}|\omega, \Theta_{\text{Expert}})dP(\omega)}, \tag{D.35}$$

$$= \frac{\prod_{i=1}^{N_{\text{Expert}}} p(\tau_i|\omega, \theta_i)p(\omega)}{\int \prod_{i=1}^{N_{\text{Expert}}} p(\tau_i|\omega, \theta_i)dP(\omega)}, \tag{D.36}$$

$$= \frac{\prod_{i=1}^{N_{\text{Expert}}} \prod_{j=0}^{H_i-1} p(s_{j+1}|s_j, a_j, \theta_i)p(a_j|s_j, \omega, \theta_i)p(\omega)}{\int \prod_{i=1}^{N_{\text{Expert}}} \prod_{j=0}^{H_i-1} p(s_{j+1}|s_j, a_j, \theta_i)p(a_j|s_j, \omega, \theta_i)dP(\omega)}, \tag{D.37}$$

$$= \frac{\prod_{i=1}^{N_{\text{Expert}}} \prod_{j=0}^{H_i-1} p(a_j|s_j, \omega, \theta_i)p(\omega)}{\int \prod_{i=1}^{N_{\text{Expert}}} \prod_{j=0}^{H_i-1} p(a_j|s_j, \omega, \theta_i)dP(\omega)}, \tag{D.38}$$

where we have used the fact that each $p(s_{j+1}|s_j, a_j, \theta_i)$ has no dependence on ω, and so will cancel in the fraction when deriving the final line. Now, substituting for the definition of the likelihood:

$$p(\omega|\mathcal{D}_{\text{Expert}}, \Theta_{\text{Expert}}) \tag{D.39}$$

$$\propto \underbrace{\exp\left(\sum_{i=1}^{N_{\text{Expert}}} \sum_{j=0}^{H_i-1} \left(\frac{\Psi^\star(s_j, a_j, \theta_i)^\top \omega}{\alpha} - \log z(s_j, \theta_i, \omega)\right)\right)}_{\text{Likelihood}} \underbrace{\exp\left(-\frac{\|\omega - \omega_0\|^2}{2\sigma_0^2}\right)}_{\text{Prior}}, \tag{D.40}$$

$$= \exp\left(\sum_{i=1}^{N_{\text{Expert}}} \sum_{j=0}^{H_i-1} \left(\frac{\Psi^\star(s_j, a_j, \theta_i)^\top \omega}{\alpha} - \log z(s_j, \theta_i, \omega)\right) - \frac{\|\omega - \omega_0\|^2}{2\sigma_0^2}\right), \tag{D.41}$$

hence:

$$\nabla_\omega \log p(\omega|\mathcal{D}_{\text{Expert}}, \Theta_{\text{Expert}}) \tag{D.42}$$

$$= \nabla_\omega \left(\sum_{i=1}^{N_{\text{Expert}}} \sum_{j=0}^{H_i-1} \left(\frac{\Psi^\star(s_j, a_j, \theta_i)^\top \omega}{\alpha} - \log z(s_j, \theta_i, \omega) \right) - \frac{\|\omega - \omega_0\|^2}{2\sigma_0^2} \right), \tag{D.43}$$

$$\implies g(\omega; \mathcal{D}_{\text{Expert}}) \tag{D.44}$$

$$= \int_{\Theta^{N_{\text{Expert}}}} \nabla_\omega \left(\sum_{i=1}^{N_{\text{Expert}}} \sum_{j=0}^{H_i-1} \left(\frac{\Psi^\star(s_j, a_j, \theta_i)^\top \omega}{\alpha} - \log z(s_j, \theta_i, \omega) \right) \right. \tag{D.45}$$

$$\left. - \frac{\|\omega - \omega_0\|^2}{2\sigma_0^2} \right) dP(\Theta_{\text{Expert}}|\omega, \mathcal{D}_{\text{Expert}}), \tag{D.46}$$

$$= \sum_{i=1}^{N_{\text{Expert}}} \mathbb{E}_{\theta_i \sim p(\theta_i|\omega,\tau_i)} \left[\nabla_\omega \left(\sum_{j=0}^{H_i-1} \left(\frac{\Psi^\star(s_j, a_j, \theta_i)^\top \omega}{\alpha} - \log z(s_j, \theta_i, \omega) \right) - \frac{\|\omega - \omega_0\|^2}{2\sigma_0^2} \right) \right]. \tag{D.47}$$

Now, applying Lemma 1 and multiplying by α yields:

$$\alpha g(\omega; \mathcal{D}_{\text{Expert}}) = \sum_{i=1}^{N_{\text{Expert}}} \sum_{j=0}^{H_i-1} \mathbb{E}_{\theta_i \sim p(\theta_i|\omega,\mathcal{D}_{\text{Expert}})} \left[\left(\Psi^\star(s_j, a_j, \theta_i) - \mathbb{E}_{a_i \sim p(a_i|s_j,\omega,\theta_i)} \left[\Psi^\star(s_j, a_i, \theta_i) \right] \right) \right] \tag{D.48}$$

$$- \frac{(\omega - \omega_0)}{\varsigma_0^2}, \tag{D.49}$$

as required □

D.4.1 Derivation of BRL Objective

Starting from the Bayesian RL objective:

$$J^\pi_{\text{Bayes}} = \mathbb{E}_{\tau_\infty \sim p^\pi_\infty(\tau_\infty)} \left[\sum_{i=0}^{\infty} \gamma^i r_i \right], \tag{D.50}$$

$$= \mathbb{E}_{h_\infty \sim p^\pi_\infty(h_\infty)} \left[\sum_{i=0}^{\infty} \gamma^i \mathbb{E}_{r_i \sim p(r_i|h_i,a_i)} \left[r_i \right] \right]. \tag{D.51}$$

Now,

$$\mathbb{E}_{r_i \sim p(r_i|h_i,a_i)}\left[r_i\right] = \mathbb{E}_{r_i \sim p^{\text{IRL+COE}}_{\text{Bayes}}(r_i|s_i,a_i)}\left[r_i\right], \tag{D.52}$$

$$= \begin{cases} \mathbb{E}_{k\sim p(k)}\left[\mathbb{E}_{r_i \sim p(r_i|s_i,a_i,k)}\left[r_i\right]\right], & s,a \in \mathcal{S}_{\text{COE}} \times \mathcal{A}_{\text{COE}}, \\ \mathbb{E}_{\omega\sim p(\omega)}\left[\mathbb{E}_{r_i \sim p(r_i|s_i,a_i,\omega)}\left[r_i\right]\right], & \text{otherwise}, \end{cases} \tag{D.53}$$

$$= \begin{cases} \mathbb{E}_{k\sim p(k)}\left[k r_{\max}\right], & s,a \in \mathcal{S}_{\text{COE}} \times \mathcal{A}_{\text{COE}}, \\ \mathbb{E}_{\omega\sim p(\omega)}\left[\nu(s_i,a_i)^\top \omega\right], & \text{otherwise}, \end{cases} \tag{D.54}$$

$$= \begin{cases} k^\star r_{\max}, & s,a \in \mathcal{S}_{\text{COE}} \times \mathcal{A}_{\text{COE}}, \\ \nu(s_i,a_i)^\top \omega^\star_{\text{Bayes}}, & \text{otherwise}, \end{cases} \tag{D.55}$$

$$= r^{\text{IRL+COE}}_{\text{Bayes}}(s_i,a_i), \tag{D.56}$$

hence:

$$J^\pi_{\text{Bayes}} = \mathbb{E}_{h_\infty \sim p^\pi_\infty(h_\infty)}\left[\sum_{i=0}^{\infty} \gamma^i r^{\text{IRL+COE}}_{\text{Bayes}}(s_i,a_i)\right], \tag{D.57}$$

$$= \mathbb{E}_{\theta_{\text{test}}\sim p(\theta_{\text{test}})}\left[\mathbb{E}_{h_\infty \sim p^\pi_\infty(h_\infty|\theta)}\left[\sum_{i=0}^{\infty} \gamma^i r^{\text{IRL+COE}}_{\text{Bayes}}(s_i,a_i)\right]\right], \tag{D.58}$$

as required.

D.5 The Role of Temperature in IRL

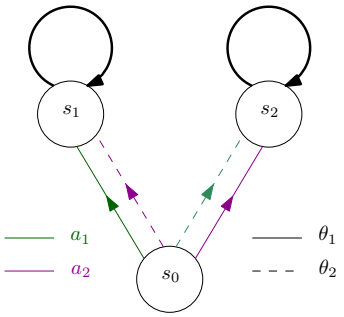


Figure D.2: Space of Three State MDPs

We now provide an intuitive example to demonstrate how the prior influences the expert data in our Bayesian formulation. Consider the space of three state MDPs in figure D.2. The agent has a set of two possible actions $\mathcal{A} = \{a_1, a_2\}$. For $\theta = \theta_1$, the agent transitions to state s_1 deterministically after selecting a_1 or s_2 after selecting a_2. For $\theta = \theta_2$, the actions are reversed. In both MDPs, the initial state is s_0 and state

s_1 or s_2 are terminal. The reward function depends only on states and is $r(s_0) = 0$, $r(s_1) = 1$, $r(s_2) = -1$. Let $\mathbb{I}(s) \in \{0,1\}^3$ denote the indicator feature vector where $I(s_n)$ is a one-hot vector where the nth element is 1, e.g. $\mathbb{I}(s_1) = (0,1,0)^\top$. We represent this reward function using the linear form $r(s) = \mathbb{I}(s)^\top \omega^\star$ where $\omega^\star = (0,1,-1)^\top$. Finally, MDPs are allocated using a uniform prior.

Expert data will always consist of trajectories that transition from state s_0 to state s_1. For our feature vector, we can derive the corresponding expert successor feature representation analytically:

$$\Psi^\star(s_0, a_1, \theta_1) = \mathbb{I}(s_0) + \frac{\gamma}{1-\gamma}\mathbb{I}(s_1), \;\; \Psi^\star(s_0, a_2, \theta_1) = \mathbb{I}(s_0) + \frac{\gamma}{1-\gamma}\mathbb{I}(s_2), \tag{D.59}$$

$$\Psi^\star(s_0, a_2, \theta_2) = \mathbb{I}(s_0) + \frac{\gamma}{1-\gamma}\mathbb{I}(s_1), \;\; \Psi^\star(s_0, a_1, \theta_2) = \mathbb{I}(s_0) + \frac{\gamma}{1-\gamma}\mathbb{I}(s_2), \tag{D.60}$$

$$\Psi^\star(s_1, \cdot, \cdot) = \frac{1}{1-\gamma}\mathbb{I}(s_1), \;\; \Psi^\star(s_2, \cdot, \cdot) = \frac{1}{1-\gamma}\mathbb{I}(s_2). \tag{D.61}$$

Using the gradient update in equation (8.3), we see that updates in state s_0 will initially draw actions equally from $p(a_i|s_0, \omega, \cdot)$. This yields an initial gradient update of:

$$g^0_\omega = \omega_0 + \eta^0_\omega \frac{N_{\text{Expert}} H_i}{2} \left(\Psi^\star(s_0, a_1, \theta_1) - \frac{1}{2}\left[\Psi^\star(s_0, a_1, \theta_1) + \Psi^\star(s_0, a_2, \theta_1)\right] \right) + \tag{D.62}$$

$$\eta_\omega \frac{N_{\text{Expert}} H_i}{2} \left(\Psi^\star(s_0, a_2, \theta_2) - \frac{1}{2}\left[\Psi^\star(s_0, a_2, \theta_2) + \Psi^\star(s_0, a_1, \theta_2)\right] \right), \tag{D.63}$$

$$= \omega_0 + \frac{\gamma \eta^0_\omega N_{\text{Expert}} H_i}{2(1-\gamma)} \left[\mathbb{I}(s_1) - \mathbb{I}(s_2)\right]. \tag{D.64}$$

For exposition, assume that there is no prior preference between ω^1_0 and ω^2_0, and that $\omega^1_0 = \omega^2_0 = 0$. We see that the initial gradient will update these values to:

$$\omega^1_1 = \frac{\gamma \eta^0_\omega N_{\text{Expert}} H_i}{2(1-\gamma)}, \quad \omega^2_1 = -\frac{\gamma \eta^0_\omega N_{\text{Expert}} H_i}{2(1-\gamma)}. \tag{D.65}$$

We now consider the regime where the temperature parameter tends to zero $\alpha \to 0$. For all future updates, as $\omega_1 > \omega_2$, the model expert policy will tend towards a deterministic function that picks the action leading to state s_1: $p(a|s_0, \omega, \theta_i) = \delta(a = a_i)$. This means that all future updates $k \geq 1$ from state s_0 will pull the

reward parametrization back to the prior value:

$$g_\omega^k = \omega_k - \frac{\eta_\omega^k(\omega_k - \omega_0)}{\varsigma_0^2}. \tag{D.66}$$

In the regime where the temperature parameter tends to infinity $\alpha \to \infty$, the model expert policy will remain uniform over all actions. Under this assumption, all future updates $k \geq 1$ from state s_0 will continue to increase the value of ω_1 and decrease the value of ω_2, whilst pulling ω_k back towards the prior in accordance with ς_0^2.

$$g_\omega^k = \omega_k + \frac{\gamma\eta_\omega^k N_{\text{Expert}} H_i}{2(1-\gamma)} \left[\mathbb{I}(s_1) - \mathbb{I}(s_2)\right] - \frac{\eta_\omega^k(\omega_k - \omega_0)}{\varsigma_0^2}. \tag{D.67}$$

When used in practice, we select α to be between 0 and ∞. Our example reveals that the smaller the temperature parameter, the less the updates will separate values of reward for states that the expert visited vs that the expert could have visited. Conversely, when α is very large, this difference will grow and can only be counteracted by the prior variance ς_0.

D.6 Experiments

D.6.1 Computer Resources

The experiments were run on servers with eight recent NVIDIA GPUs (3080, A4500, or A5000). The random seeds were run in parallel. The experiments for the gridworld were the longest and took approximately an hour to run.

D.6.2 Latent Inference Investigation Details

figure D.3 shows the learned rewards in the latent inference experiment described in section 8.4.2. The learned rewards are somewhat hard to interpret because in addition to the scaling and shifting information being lost in IRL, there is no pressure for the algorithms to keep the reward vectors sparse. Instead, they are only trying to optimize equation (8.3). Nevertheless, the rewards learned with latent inference have the same ordering between the critical states s_1, s_2, and s_3 as the ground truth reward. In contrast, the rewards learned with naive IRL do

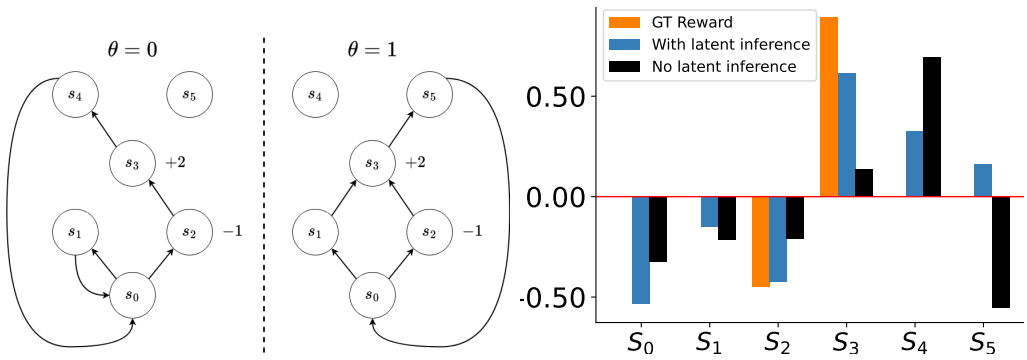


Figure D.3: A demonstration of the necessity for latent inference. On the left, we repeat the visualization of the CMDP used in the experiments for convenience. On the right, we show the rewards learned with and without latent inference compared to the ground truth rewards. The reward vectors are normalized to unit norm. The y-axis is linearly scaled.

not differentiate between s_1 and s_2, which results in the learned policies not taking the path through s_1 even when it is available.

D.6.3 Tiger Treasure Details

The Tiger-Treasure problem is displayed in figure 8.1 and described in section 8.2.1. The agent starts in state S_0 and can either listen or open a door. The listening action transitions the agent to a 'hint' state revealing with probability $p = 0.85$ the location of the Tiger. After the agent opens a door, it is either roared by the tiger or collects the treasure and arrives in the terminal state S_T. For learning the contextual IRL reward, we use the hyperparameters presented in Table D.2 with a default value of $\omega_0 = (0, 0, 0, 0, 0, 0)$ in the Laplace approximation Equation 8.3. The reward prior is introduced after having rescaled the IRL reward between $r_{\text{min}} = -100$ and $r_{\text{max}} = 10$. To enhance the training process, we normalize the rewards before training the DQN policies. The hyperparameters presented in Table D.3.

D.6.4 Tiger Treasure Maze Details

The agent starts in the middle of the two doors. It can choose to move in the cardinal directions or listen. After the agent opens a door, it will either collect a treasure or be roared at by a tiger for one timestep. Then, depending on what happened, it gets transported to the nearest grid cell to the *Tiger* or *Gold*. The

Parameter	**Value**
Number of parallel environments	500
Number of steps per rollout	50
Number of updates	5000
ϵ-greedy ϵ	0.5
Maximum gradient norm	0.5
Discount γ	0.99
α	0.01
ς_0^2	100.0
Target SF update rate	50
SF learning rate	1×10^{-3}
Reward learning rate	1×10^{-2}
Replay buffer size	50000
Batch size (trajectories)	500
$r_{\max}$	10.0
$r_{\min}$	-100.0

Table D.2: IRL Hyperparameters for the Tiger Treasure problem.

Parameter	**Value**
Number of parallel environments	16
Number of steps per rollout	50
Total number of updates	20000
Learning rate	1×10^{-4}
Discount γ	0.99
Target network update rate	1
Replay buffer size (number of full trajectories)	200000
Batch size (number of full trajectories)	100
Starting value for ϵ	1.0
Final value for ϵ	0.05
Fraction of updates after ϵ schedule finishes	0.5

Table D.3: DQN Hyperparameters for the Tiger Treasure problem.

expert takes the shortest path to *Gold* and never listens. The state of the agent is defined as the $X - Y$ coordinates of the agent and an indicator variable, which indicates the result of the listening action. After taking a listening action, the agent is transported to a state with the same $X - Y$ coordinates but with the indicator showing the true value of θ. The indicator states have the same dynamics as the corresponding normal states. The coordinates and the indicator are encoded as one-hot vectors. The reward feature ν is the full table of all states visitable by the agent. Since the dynamics are deterministic given θ, the inference model labels

Parameter	**Value**
Number of parallel environments	16
Number of steps per rollout	40
Number of updates	20000
ϵ-greedy ϵ	0.5
Maximum gradient norm	0.5
Discount γ	0.99
α	0.01
ς_0^2	1.0
Target SF update rate	50
SF learning rate	3×10^{-4}
Reward learning rate	3×10^{-3}
Replay buffer size (number of full trajectories)	10000
Batch size (number of full trajectories)	100
k	0.01
$r_{\max}$	1.0
$r_{\min}$	-0.05

Table D.4: IRL Hyperparameters for the Maze problem.

Parameter	**Value**
Number of parallel environments	16
Number of steps per rollout	40
Total number of updates	100000
Learning rate	1×10^{-4}
Discount γ	0.99
Target network update rate	1
Replay buffer size (number of full trajectories)	10000
Batch size (number of full trajectories)	100
Starting value for ϵ	1.0
Final value for ϵ	0.05
Fraction of updates after ϵ schedule finishes	0.5

Table D.5: DQN Hyperparameters for the Maze problem.

the trajectories with the true θ if it is revealed on the trajectory.

The hyperparameters used for BIG in the maze experiment are shown in Table D.4. The hyperparameters for DQN in the maze experiment are shown in Table D.5. A recurrent neural network is used for the policy architecture. A separate MLP network is used for implementing the critic.

References

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. *Advances in Neural Information Processing Systems*, 34:7799–7812, 2021.

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

S. Adams, T. Cody, and P. A. Beling. A survey of inverse reinforcement learning. *Artif. Intell. Rev.*, 55(6):4307–4346, aug 2022. ISSN 0269-2821. doi: 10.1007/s10462-021-10108-x. URL `https://doi.org/10.1007/s10462-021-10108-x`.

M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017.

F. Alet, M. F. Schneider, T. Lozano-Perez, and L. P. Kaelbling. Meta-learning curiosity algorithms. In *International Conference on Learning Representations*, 2020.

D. Almeida, C. Winter, J. Tang, and W. Zaremba. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, pages 13544–13555, 2019.

S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103500. URL `https://www.sciencedirect.com/science/article/pii/S0004370221000515`.

K. J. Åström. Optimal control of markov processes with incomplete state information i. *Journal of mathematical analysis and applications*, 10:174–205, 1965.

A. Aubret, L. Matignon, and S. Hassas. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*, 2019.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/350db081a661525235354dd3e19b8c05-Paper.pdf`.

S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE Computer Society, 2021.

J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

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

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

H. Bharadhwaj, J. Vakil, M. Sharma, A. Gupta, S. Tulsiani, and V. Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4788–4795. IEEE, 2024.

C. Bonnet, P. Caron, T. Barrett, I. Davies, and A. Laterre. One step at a time: Pros and cons of multi-step meta-gradient reinforcement learning. *arXiv preprint arXiv:2111.00206*, 2021.

K. Bousmalis, G. Vezzani, D. Rao, C. Devin, A. X. Lee, M. Bauza, T. Davchev, Y. Zhou, A. Gupta, A. Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

E. Bronstein, M. Palatucci, D. Notz, B. White, A. Kuefler, Y. Lu, S. Paul, P. Nikdel, P. Mougin, H. Chen, J. Fu, A. Abrams, P. Shah, E. Racah, B. Frenkel, S. Whiteson, and D. Anguelov. Hierarchical model-based imitation learning for planning in autonomous driving, 2022. URL `https://arxiv.org/abs/2210.09539`.

D. S. Brown and S. Niekum. Deep bayesian reward learning from preferences. *Workshop on Safety and Robustness in Decision Making. 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, abs/1912.04472, 2019. URL `https://arxiv.org/pdf/1912.04472.pdf`.

T. B. Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

B. Burega, J. D. Martin, and M. Bowling. Learning to prioritize planning updates in model-based reinforcement learning. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=uR7ePjeB6z`.

X.-Q. Cai, Y.-X. Ding, Z.-X. Chen, Y. Jiang, M. Sugiyama, and Z.-H. Zhou. Seeing differently, acting similarly: Heterogeneously observable imitation learning. *arXiv preprint arXiv:2106.09256*, 2021.

A. S. Chen, S. Nair, and C. Finn. Learning generalizable robotic reward functions from" in-the-wild" human videos. *arXiv preprint arXiv:2103.16817*, 2021.

D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Conference on Robot Learning (CoRL)*, 2019.

Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pages 748–756. PMLR, 2017.

C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

J. D. Co-Reyes, Y. Miao, D. Peng, E. Real, Q. V. Le, S. Levine, H. Lee, and A. Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=0XXpJ4OtjW`.

F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9329–9338, 2019.

E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

Z. J. Cui, Y. Wang, N. M. M. Shafiullah, and L. Pinto. From play to policy: Conditional behavior generation from uncurated robot data. *arXiv preprint arXiv:2210.10047*, 2022.

N. Das, S. Bechtle, T. Davchev, D. Jayaraman, A. Rai, and F. Meier. Model-based inverse reinforcement learning from visual demonstrations. In *Conference on Robot Learning*, pages 1930–1942. PMLR, 2021.

S. Dasari and A. Gupta. Transformers for one-shot visual imitation. In *Conference on Robot Learning*, pages 2071–2084. PMLR, 2021.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

P. de Haan, D. Jayaraman, and S. Levine. Causal confusion in imitation learning. *Advances in Neural Information Processing Systems*, 32, 2019.

T. Deleu and Y. Bengio. The effects of negative adaptation in model-agnostic meta-learning. *arXiv preprint arXiv:1812.02159*, 2018.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

J. L. Doob. Application of the theory of martingales. *Colloques Internationaux du Centre National de la Recherche Scientifique*, pages 23–27, 1949.

Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Y. Duan, M. Andrychowicz, B. Stadie, O. Jonathan Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 30, 2017.

M. O. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

K. Dvijotham and E. Todorov. Inverse optimal control with linearly-solvable mdps. In *Proceedings of the 27th International conference on machine learning (ICML-10)*, pages 335–342, 2010.

A. D. Edwards and C. L. Isbell. Perceptual values from observation. *arXiv preprint arXiv:1905.07861*, 2019.

P. Englert and M. Toussaint. Learning manipulation skills from a single demonstration. *The International Journal of Robotics Research*, 37(1):137–154, 2018.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

A. Fallah, K. Georgiev, A. Mokhtari, and A. Ozdaglar. Provably convergent policy gradient methods for model-agnostic meta-reinforcement learning. *arXiv preprint arXiv:2002.05135*, 2020.

A. Fallah, K. Georgiev, A. Mokhtari, and A. Ozdaglar. On the convergence theory of debiased model-agnostic meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 34:3096–3107, 2021.

H.-S. Fang, H. Fang, Z. Tang, J. Liu, J. Wang, H. Zhu, and C. Lu. Rh20t: A robotic dataset for learning diverse skills in one-shot. In *RSS 2023 Workshop on Learning for Task and Motion Planning*, 2023.

G. Farquhar, S. Whiteson, and J. Foerster. Loaded dice: Trading off bias and variance in any-order score function gradient estimators for reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

M. Fellows, B. Kaplowitz, C. S. de Witt, and S. Whiteson. Bayesian exploration networks. *arXiv preprint arXiv:2308.13049*, 2023a.

M. Fellows, M. J. A. Smith, and S. Whiteson. Why target networks stabilise temporal difference methods. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 9886–9909. PMLR, 23–29 Jul 2023b. URL `https://proceedings.mlr.press/v202/fellows23a.html`.

A. Filos, C. Lyle, Y. Gal, S. Levine, N. Jaques, and G. Farquhar. Psiphi-learning: Reinforcement learning with demonstrations using successor features and inverse temporal difference learning, 2021.

C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017a.

C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017b.

S. Flennerhag, A. A. Rusu, R. Pascanu, F. Visin, H. Yin, and R. Hadsell. Meta-learning with warped gradient descent. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkeiQlBFPB`.

S. Flennerhag, Y. Schroecker, T. Zahavy, H. van Hasselt, D. Silver, and S. Singh. Bootstrapped meta-learning. *arXiv preprint arXiv:2109.04504*, 2021.

P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

J. Foerster, G. Farquhar, M. Al-Shedivat, T. Rocktäschel, E. Xing, and S. Whiteson. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018.

L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173. PMLR, 2017.

K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. META LEARNING SHARED HIERARCHIES. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyX0IeWAW`.

H. Fu, H. Tang, J. Hao, W. Liu, and C. Chen. Mghrl: Meta goal-generation for hierarchical reinforcement learning. In *International Conference on Distributed Artificial Intelligence*, pages 29–39. Springer, 2020.

J. Fu, K. Luo, and S. Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, 2024.

K. Gao and O. Sener. Modeling and optimization trade-off in meta-learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11154–11165. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/7fc63ff01769c4fa7d9279e97e307829-Paper.pdf`.

J. J. Garau-Luis, Y. Miao, J. D. Co-Reyes, A. Parisi, J. Tan, E. Real, and A. Faust. Multi-objective evolution for generalizable policy gradient algorithms. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022.

D. Garg, S. Chakraborty, C. Cundy, J. Song, and S. Ermon. Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, 34: 4028–4039, 2021.

A. Ghadirzadeh, X. Chen, P. Poklukar, C. Finn, M. Björkman, and D. Kragic. Bayesian meta-learning for few-shot policy adaptation across robotic platforms. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1274–1280, 2021. doi: 10.1109/IROS51168.2021.9636628.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/4de754248c196c85ee4fbdcee89179bd-Paper.pdf`.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *"International Conference on Learning Representations"*, 2020.

A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

A. Harutyunyan, W. Dabney, T. Mesnard, M. G. Azar, B. Piot, N. Heess, H. P. van Hasselt, G. Wayne, S. Singh, D. Precup, et al. Hindsight credit assignment. In *Advances in neural information processing systems*, pages 12467–12476, 2019.

E. Hazan, S. Kakade, K. Singh, and A. Van Soest. Provably efficient maximum entropy exploration. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2681–2691. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/hazan19a.html`.

I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR (Poster)*, 3, 2017.

J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel. Evolved policy gradients. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/7876acb66640bad41f1e1371ef30c180-Paper.pdf`.

M. Huisman, J. N. Van Rijn, and A. Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541, 2021.

C.-C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):1–12, 2019.

M. T. Jackson, M. Jiang, J. Parker-Holder, R. Vuorio, C. Lu, G. Farquhar, S. Whiteson, and J. N. Foerster. Discovering general reinforcement learning algorithms with adversarial environment design. *arXiv preprint arXiv:2310.02782*, 2023.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

S. James, M. Bloesch, and A. J. Davison. Task-embedded control networks for few-shot imitation learning. In *Conference on robot learning*, pages 783–795. PMLR, 2018.

E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 991–1002. PMLR, 08–11 Nov 2022. URL `https://proceedings.mlr.press/v164/jang22a.html`.

D. Janz, J. Hron, P. Mazur, K. Hofmann, J. M. Hernández-Lobato, and S. Tschiatschek. Successor uncertainties: Exploration and uncertainty in temporal difference learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630, May 1957. doi: 10.1103/PhysRev.106.620. URL `https://link.aps.org/doi/10.1103/PhysRev.106.620`.

E. Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 4613–4619. IEEE, 2021.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

R. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, D*, 82:35–44, 1960.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

L. Kirsch, S. Flennerhag, H. van Hasselt, A. Friesen, J. Oh, and Y. Chen. Introducing symmetries to black box meta reinforcement learning. *arXiv preprint arXiv:2109.10781*, 2021.

V. Kolev, R. Rafailov, K. Hatch, J. Wu, and C. Finn. Efficient imitation learning with conservative world models. *arXiv preprint arXiv:2405.13193*, 2024.

J. Z. Kolter, M. P. Rodgers, and A. Y. Ng. A control architecture for quadruped locomotion over rough terrain. In *2008 IEEE International Conference on Robotics and Automation*, pages 811–818. IEEE, 2008.

R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

D. Kumor, J. Zhang, and E. Bareinboim. Sequential causal imitation learning with unobserved confounders. *Advances in Neural Information Processing Systems*, 34: 14669–14680, 2021.

M. Kwon, S. Daptardar, P. R. Schrater, and X. Pitkow. Inverse rational control with partially observable continuous nonlinear dynamics. *Advances in neural information processing systems*, 33:7898–7909, 2020.

L. Lai, A. Z. Huang, and S. J. Gershman. Action chunking as policy compression, 2022.

Q. Lan, A. R. Mahmood, S. Yan, and Z. Xu. Learning to optimize for reinforcement learning. *arXiv preprint arXiv:2302.01470*, 2023.

R. T. Lange. evosax: Jax-based evolution strategies, 2022. URL `http://github.com/RobertTLange/evosax`.

M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pages 143–156. PMLR, 2017.

J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

K. Li and J. Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few shot learning. *CoRR*, abs/1707.09835, 2017. URL `http://arxiv.org/abs/1707.09835`.

R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016. URL `http://arxiv.org/abs/1509.02971`.

X. Lin, Harjatin, S. Baweja, G. Kantor, and D. Held. Adaptive auxiliary task weighting for reinforcement learning. In *NeurIPS*, 2019.

B. Liu, X. Feng, H. Zhang, J. Wang, and Y. Yang. Settling the bias and variance of meta-gradient estimation for meta-reinforcement learning. *arXiv preprint arXiv:2112.15400*, 2021.

B. Liu, X. Feng, J. Ren, L. Mai, R. Zhu, H. Zhang, J. Wang, and Y. Yang. A theoretical understanding of gradient bias in meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 35:31059–31072, 2022.

H. Liu, R. Socher, and C. Xiong. Taming maml: Efficient unbiased meta-reinforcement learning. In *International Conference on Machine Learning*, pages 4061–4071. PMLR, 2019.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

B. Lou, N. Zhao, and J. Wang. Meta-learning from sparse recovery. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

C. Lu, J. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, and J. Foerster. Discovered policy optimisation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 16455–16468. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/688c7a82e31653e7c256c6c29fd3b438-Paper-Conference.pdf`.

J. Luketina, S. Flennerhag, Y. Schroecker, D. Abel, T. Zahavy, and S. Singh. Meta-gradients in non-stationary environments. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022. URL `https://openreview.net/forum?id=SlzBXwZIZ9`.

C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

J. Mao, J. Foerster, T. Rocktäschel, M. Al-Shedivat, G. Farquhar, and S. Whiteson. A baseline for any order gradient estimation in stochastic computation graphs. In *International Conference on Machine Learning*, pages 4343–4351. PMLR, 2019.

P. Marbach and J. N. Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

R. Meier and A. Mujika. Open-ended reinforcement learning with neural reward functions. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=NL05_JGVg99`.

R. Mendonca, A. Gupta, R. Kralev, P. Abbeel, S. Levine, and C. Finn. Guided meta-policy search. *NeurIPS*, 2019.

T. Mesnard, T. Weber, F. Viola, S. Thakoor, A. Saade, A. Harutyunyan, W. Dabney, T. Stepleton, N. Heess, A. Guez, et al. Counterfactual credit assignment in model-free reinforcement learning. *arXiv preprint arXiv:2011.09464*, 2020.

L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565. PMLR, 2019.

L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

E. Mitchell, R. Rafailov, X. B. Peng, S. Levine, and C. Finn. Offline meta-reinforcement learning with advantage weighting. *CoRR*, abs/2008.06043, 2020. URL `https://arxiv.org/abs/2008.06043`.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

K. P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013. ISBN 9780262018029 0262018020.

O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

T. Nam, S.-H. Sun, K. Pertsch, S. J. Hwang, and J. J. Lim. Skill-based meta-reinforcement learning. In *International Conference on Learning Representations*, 2022.

Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

A. Y. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX: The 9th international symposium on experimental robotics*, pages 363–372. Springer, 2006.

H. H. Nguyen, A. Baisero, D. Wang, C. Amato, and R. Platt. Leveraging fully observable policies for learning under partial observability. In *6th Annual Conference on Robot Learning*, 2022.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

P. Ortega and D. Braun. A bayesian rule for adaptive control based on causal interventions. In *Third Conference on Artificial General Intelligence*, 2010a.

P. A. Ortega and D. A. Braun. A minimum relative entropy principle for learning and acting. *Journal of Artificial Intelligence Research*, 38:475–511, 2010b.

P. A. Ortega, M. Kunesch, G. Delétang, T. Genewein, J. Grau-Moya, J. Veness, J. Buchli, J. Degrave, B. Piot, J. Perolat, et al. Shaking the foundations: delusions in sequence models for interaction and control. *arXiv preprint arXiv:2110.10819*, 2021.

I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepezvari, S. Singh, et al. Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568*, 2019.

A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

E. Park and J. B. Oliva. Meta-curvature. *NeurIPS*, 2019.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

J. Pearl. *Causality*. Cambridge University Press, Sept. 2009. URL `https://www.cambridge.org/core/books/causality/B0046844FAE10CBF274D4ACBDAEB5F5B`.

E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL `https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf`.

I. Radosavovic, T. Xiao, S. James, P. Abbeel, J. Malik, and T. Darrell. Real-world robot learning with masked visual pre-training. In *Conference on Robot Learning*, pages 416–426. PMLR, 2023.

R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*, abs/2305.18290, 2023. URL `https://api.semanticscholar.org/CorpusID:258959321`.

A. Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of MAML. *ICLR*, 2020.

R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE, 2018.

J. Rajendran, R. Lewis, V. Veeriah, H. Lee, and S. Singh. How should an agent practice? *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5454–5461, Apr. 2020. doi: 10.1609/aaai.v34i04.5995. URL `https://ojs.aaai.org/index.php/AAAI/article/view/5995`.

D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2586–2591, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

D. Raposo, S. Ritter, A. Santoro, G. Wayne, T. Weber, M. Botvinick, H. van Hasselt, and F. Song. Synthetic returns for long-term credit assignment. *arXiv preprint arXiv:2102.12425*, 2021.

N. D. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27:25–53, 2009.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Dr.-Ing.* PhD thesis, Thesis, Technical University of Berlin, Department of Process Engineering, 1971.

S. Reddy, A. D. Dragan, and S. Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

A. Z. Ren, B. Govil, T.-Y. Yang, K. R. Narasimhan, and A. Majumdar. Leveraging language for accelerated learning of tool manipulation. In *6th Annual Conference on Robot Learning*, 2022. URL `https://openreview.net/forum?id=nPw7jaGBrCG`.

J. Ren, G. Swamy, Z. S. Wu, J. A. Bagnell, and S. Choudhury. Hybrid inverse reinforcement learning. *arXiv preprint arXiv:2402.08848*, 2024.

D. J. Rezende, I. Danihelka, G. Papamakarios, N. R. Ke, R. Jiang, T. Weber, K. Gregor, H. Merzic, F. Viola, J. Wang, et al. Causally correct partial models for reinforcement learning. *arXiv preprint arXiv:2002.02836*, 2020.

E. Rosete-Beas, O. Mees, G. Kalweit, J. Boedecker, and W. Burgard. Latent plans for task-agnostic offline reinforcement learning. In *Conference on Robot Learning*, pages 1838–1849. PMLR, 2023.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

K. Ruan, J. Zhang, X. Di, and E. Bareinboim. Causal imitation learning via inverse reinforcement learning. In *International Conference on Learning Representations*, 2022.

S. Russell. Learning agents for uncertain environments (extended abstract). In *The Eleventh Annual Conference on Computational Learning Theory*, 1998.

C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

H.-P. Schwefel. Evolutionsstrategien für die numerische optimierung. In *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, pages 123–176. Springer, 1977.

N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35: 22955–22968, 2022.

L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg. Concept2robot: Learning manipulation concepts from instructions and human demonstrations. *The International Journal of Robotics Research*, 40(12-14):1419–1434, 2021.

M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR, 2022.

M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023.

D. Silver, J. A. Bagnell, and A. Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

S. Singh, R. L. Lewis, and A. G. Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.

L. Smith, N. Dhawan, M. Zhang, P. Abbeel, and S. Levine. Avid: Learning multi-stage tasks via pixel-level translation of human videos. *arXiv preprint arXiv:1912.04443*, 2019.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

X. Song, W. Gao, Y. Yang, K. Choromanski, A. Pacchiano, and Y. Tang. Es-maml: Simple hessian-free meta learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1exA2NtDB`.

J. Spencer, S. Choudhury, A. Venkatraman, B. Ziebart, and J. A. Bagnell. Feedback in imitation learning: The three regimes of covariate shift. *arXiv preprint arXiv:2102.02872*, 2021.

M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons, 2020.

B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

F. Sung, L. Zhang, T. Xiang, T. M. Hospedales, and Y. Yang. Learning to learn: Meta-critic networks for sample efficient learning. *CoRR*, abs/1706.09529, 2017. URL `http://arxiv.org/abs/1706.09529`.

I. Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

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

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

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

G. Swamy, S. Choudhury, J. A. Bagnell, and S. Wu. Of moments and matching: A game-theoretic framework for closing the imitation gap. In *International Conference on Machine Learning*, pages 10022–10032. PMLR, 2021.

G. Swamy, S. Choudhury, J. A. Bagnell, and Z. S. Wu. Causal imitation learning under temporally correlated noise, 2022a.

G. Swamy, S. Choudhury, J. A. Bagnell, and Z. S. Wu. Sequence model imitation learning with unobserved contexts. *Advances in Neural Information Processing Systems*, 35, 2022b.

J. Tack, J. Park, H. Lee, J. Lee, and J. Shin. Meta-learning with self-improving momentum target. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=FCNMbF_TsKm`.

Y. Tang. Biased gradient estimate with drastic variance reduction for meta reinforcement learning. *arXiv preprint arXiv:2112.07328*, 2021.

Y. Tang. Biased gradient estimate with drastic variance reduction for meta reinforcement learning. In *International Conference on Machine Learning*, pages 21050–21075. PMLR, 2022.

Y. Tang, T. Kozuno, M. Rowland, R. Munos, and M. Valko. Unifying gradient estimators for meta-reinforcement learning via off-policy evaluation. *arXiv preprint arXiv:2106.13125*, 2021.

O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

S. Thrun and L. Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997. doi: 10.1109/9.580874.

E. Valassakis, G. Papagiannis, N. Di Palo, and E. Johns. Demonstrate once, imitate immediately (dome): Learning visual servoing for one-shot imitation learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8614–8621. IEEE, 2022.

A. van der Vaart. *Asymptotic Statistics*. Cambridge series on statistical and probabilistic mathematics. Cambridge University Press, 1998. ISBN 9780521496032. URL `https://books.google.co.uk/books?id=fiX9ngEACAAJ`.

H. van Hasselt, S. Madjiheurem, M. Hessel, D. Silver, A. Barreto, and D. Borsa. Expected eligibility traces. *arXiv preprint arXiv:2007.01839*, 2020.

J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

V. Veeriah, M. Hessel, Z. Xu, J. Rajendran, R. L. Lewis, J. Oh, H. P. van Hasselt, D. Silver, and S. Singh. Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, pages 9306–9317, 2019.

V. Veeriah, T. Zahavy, M. Hessel, Z. Xu, J. Oh, I. Kemaev, H. van Hasselt, D. Silver, and S. Singh. Discovery of options via meta-learned subgoals. *arXiv preprint arXiv:2102.06741*, 2021.

R. Vuorio, S.-H. Sun, H. Hu, and J. Lim. Multimodal model-agnostic meta-learning via task-aware modulation. In *NeurIPS*, 2019.

R. Vuorio, J. A. Beck, G. Farquhar, J. N. Foerster, and S. Whiteson. No dice: An investigation of the bias-variance tradeoff in meta-gradients. In *Deep RL Workshop NeurIPS 2021*, 2021.

R. Vuorio, M. Fellows, C. Lu, C. Grislain, and S. Whiteson. A bayesian solution to the imitation gap, 2024a. URL `https://arxiv.org/abs/2407.00495`.

R. Vuorio, P. D. Haan, J. Brehmer, H. Ackermann, D. Dijkman, and T. Cohen. Deconfounding imitation learning with variational inference. *Transactions on Machine Learning Research*, 2024b. ISSN 2835-8856. URL `https://openreview.net/forum?id=3FsVtsISHW`. Expert Certification.

A. Walsman, M. Zhang, S. Choudhury, D. Fox, and A. Farhadi. Impossibly good experts and how to follow them. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=sciA_xgYofB`.

C. Wang, L. Fan, J. Sun, R. Zhang, L. Fei-Fei, D. Xu, Y. Zhu, and A. Anandkumar. Mimicplay: Long-horizon imitation learning by watching human play. *arXiv preprint arXiv:2302.12422*, 2023.

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. In *Annual Meeting of the Cognitive Science Society*, 2016.

R. Wang, C. Ciliberto, P. V. Amadori, and Y. Demiris. Random expert distillation: Imitation learning via expert policy support estimation. In *International Conference on Machine Learning*, pages 6536–6544. PMLR, 2019a.

Y. Wang, Q. Ye, and T.-Y. Liu. Beyond exponentially discounted sum: Automatic learning of return function. *arXiv preprint arXiv:1905.11591*, 2019b.

Z. Wang, Y. Zhao, P. Yu, R. Zhang, and C. Chen. Bayesian meta sampling for fast uncertainty adaptation. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Bkxv90EKPB`.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Mach. Learn.*, 8(3):279–292, May 1992. URL `https://doi.org/10.1007/BF00992698`.

L. Weihs, U. Jain, I.-J. Liu, J. Salvador, S. Lazebnik, A. Kembhavi, and A. Schwing. Bridging the imitation gap by adaptive insubordination. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=WlxODqiUTD_`.

C. Wen, J. Lin, T. Darrell, D. Jayaraman, and Y. Gao. Fighting copycat agents in behavioral cloning from observation histories. *Advances in Neural Information Processing Systems*, 33:2564–2575, 2020.

C. Wen, J. Qian, J. Lin, J. Teng, D. Jayaraman, and Y. Gao. Fighting fire with fire: avoiding dnn shortcuts through priming. In *International Conference on Machine Learning*, pages 23723–23750. PMLR, 2022.

L. Weng. Policy gradient algorithms. *lilianweng.github.io*, 2018. URL `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`.

D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

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

Y. Wu, M. Ren, R. Liao, and R. Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

H. Xiong, Q. Li, Y.-C. Chen, H. Bharadhwaj, S. Sinha, and A. Garg. Learning by watching: Physical imitation of manipulation skills from human videos. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7827–7834. IEEE, 2021.

Z. Xu, H. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. *arXiv preprint arXiv:1805.09801*, 2018.

Z. Xu, H. van Hasselt, M. Hessel, J. Oh, S. Singh, and D. Silver. Meta-gradient reinforcement learning with an objective discovered online. *arXiv preprint arXiv:2007.08433*, 2020.

J. Yang, D. Sadigh, and C. Finn. Polybot: Training one policy across robots while embracing variability. *arXiv preprint arXiv:2307.03719*, 2023.

J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/e1021d43911ca2c1845910d84f40aeae-Paper.pdf`.

T. Yu, C. Finn, S. Dasari, A. Xie, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, 2018a. doi: 10.15607/RSS.2018.XIV.002.

T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018b.

T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. van Hasselt, D. Silver, and S. Singh. A self-tuning actor-critic algorithm. *arXiv preprint arXiv:2002.12928*, 2020.

T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Z. Zheng, J. Oh, and S. Singh. On learning intrinsic rewards for policy gradient methods. *arXiv preprint arXiv:1804.06459*, 2018.

Z. Zheng, J. Oh, M. Hessel, Z. Xu, M. Kroiss, H. Van Hasselt, D. Silver, and S. Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.

Z. Zheng, R. Vuorio, R. Lewis, and S. Singh. Pairwise weights for temporal credit assignment. *arXiv preprint arXiv:2102.04999*, 2021.

W. Zhou, L. Pinto, and A. Gupta. Environment probing interaction policies. In *International Conference on Learning Representations*, 2019.

B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008a.

B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 322–331, 2008b.

L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2020.

L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. *ICLR*, 2019.

L. M. Zintgraf, L. Feng, C. Lu, M. Igl, K. Hartikainen, K. Hofmann, and S. Whiteson. Exploration in approximate hyper-state space for meta reinforcement learning. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12991–13001. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/zintgraf21a.html`.

H. Zou, T. Ren, D. Yan, H. Su, and J. Zhu. Learning task-distribution reward shaping with meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11210–11218, 2021.

Y. Zou and X. Lu. Gradient-em bayesian meta-learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20865–20875. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/ef48e3ef07e359006f7869b04fa07f5e-Paper.pdf`.

M. Zucker, N. Ratliff, M. Stolle, J. Chestnutt, J. A. Bagnell, C. G. Atkeson, and J. Kuffner. Optimization and learning for rough terrain legged locomotion. *The International Journal of Robotics Research*, 30(2):175–191, 2011.