New Frameworks for Structured Policy Learning

Thesis by
Hoang Minh Le

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2020
Defended October 22, 2019

Hoang Minh Le
ORCID: 0000-0002-5521-5856

ACKNOWLEDGEMENTS

First and foremost, I am deeply grateful for having the guidance of my Ph.D. advisor, Yisong Yue. This dissertation would not have been possible without his support and invaluable insights. I have learned a great deal from Yisong over the past 5 years. I have many fond memories of my first conference talk, my first paper submission, my first research proposal all with Yisong generously spending late night hours to help me take the first steps towards being a machine learning researcher. Outside of technical matters, Yisong's kindness, patience, and encouragement have been an inspiration. Yisong, it has been an honor and privilege to be your Ph.D. student.

I would like to thank my Ph.D. committee members, Adam Wierman, Yisong Yue, Anima Anandkumar, and Hal Daumé, for taking the time out of their packed schedules to provide valuable comments and suggestions.

I managed to get "research getaways" at Microsoft Research NYC and Disney Research during my Ph.D. years as summer research interns. I would like to thank my collaborators and mentors, Peter Carr, Alekh Agarwal, Nan Jiang, Miro Dudík, Hal Daumé and Rob Schapire for teaching me new research perspectives. I had a fantastic exposure to the diverse research environment at Disney Research lab in Pittsburgh. I was continuously star-struck around the floor of MSR office in NYC, talking to researchers whose work I greatly admired. Both internships were highly rewarding and eye-opening experiences.

I also want to thank other collaborators (in no particular order): Cameron Voloshin, Abhinav Verma, Swarat Chaudhuri, Luciana Cendon, Victor Dorobantu, Andrew Taylor, Aaron Ames, Andrew Kang, Jianhui Chen and Patrick Lucey. I have learned tremendously thanks to having such amazing people work with. It has also been fun discussing many ideas and problems with my lab members, Stephan Zheng, Jialin Song, Eric Zhan, Yuxin Chen, Yanan Sui, Joe Marino, Guanya Shi, Angie Liu, Rose Yu, Jeremy Bernstein, Ellen Feldman, and Taehwan Kim.

The Department of Computing and Mathematical Sciences at Caltech has given me amazing research environment and access to inspiring faculty members. Coming from previous non-academic work experience, it was invaluable to take the courses from Joel Tropp, Venkat Chandrasekaran, Adam Wierman, Yisong Yue, Houman Owhadi, Thomas Vidick as a first year graduate student.

I want to acknowledge my other fellow Caltech students, who made Ph.D. life fun

and memorable. Thanks Utkan, Corina, Dimitar, Nina, Benson, Sahin, Oguzhan, Halime, Alvita, Dawna, Kyu, Jing, Andrea, and Jenish for many fun nights and laughters. I want to give a shout-out to my Vietnamese soccer team and support group at Caltech, Kien & Phuong, Lam & Uyen, Tri & Tran, Phong & Duong, anh Thang & chi Nhan. I am very grateful to have all of you guys as part of my Caltech family.

And to my Caltech home, I am extremely lucky to have my partner Zeynep and my puppy Tobias with me through the Ph.D. journey. You guys have kept me sane, healthy, well-fed, entertained and motivated along the way. None of this would have been possible without you. Thank you for being my home!

Finally and most lovingly, I thank my immediate family for their endless love and sacrifice. I am greatly indebted to my loving parents, who did everything to give me amazing opportunities, despite how little they had. It is through their constant support and understanding that I have had such a rich life experience. I thank my sister and my little niece Trang for always being there.

ABSTRACT

Sequential decision making applications are playing an increasingly important role in everyday life. Research interest in machine learning approaches to sequential decision making has surged thanks to recent empirical successes of reinforcement learning and imitation learning techniques, partly fueled by recent advances in deep learning-based function approximation. However in many real-world sequential decision making applications, relying purely on black box policy learning is often insufficient, due to practical requirements of data efficiency, interpretability, safety guarantees, etc. These challenges collectively make it difficult for many existing policy learning methods to find success in realistic applications.

In this dissertation, we present recent advances in *structured policy learning*, which are new machine learning frameworks that integrate policy learning with principled notions of domain knowledge, which spans value-based, policy-based, and model-based structures. Our framework takes flexible reduction-style approaches that can integrate structure with reinforcement learning, imitation learning and robust control techniques. In addition to methodological advances, we demonstrate several successful applications of the new policy learning frameworks.

PUBLISHED CONTENT AND CONTRIBUTIONS

Jianhui Chen, Hoang M Le, Peter Carr, Yisong Yue, and James J Little. Learning online smooth predictors for realtime camera planning using recurrent decision trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4688–4696, 2016. URL http://openaccess.thecvf.com/content_cvpr_2016/html/Chen_Learning_Online_Smooth_CVPR_2016_paper.html.

H.M.L participated in developing the algorithm, implemented and analyzed the method, conducted the experiments, and participated in the writing of the manuscript.

Hoang Le, Andrew Kang, Yisong Yue, and Peter Carr. Smooth imitation learning for online sequence prediction. In *International Conference on Machine Learning*, pages 680–688, 2016. URL http://proceedings.mlr.press/v48/le16.html.

H.M.L participated in the formulation of the project, performed theoretical analysis, implemented and analyzed the method, conducted the experiments, and participated in the writing of the manuscript.

Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, pages 2923–2932, 2018. URL http://proceedings.mlr.press/v80/le18a.html.

H.M.L participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted the experiments, and participated in the writing of the manuscript.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712, 2019. URL http://proceedings.mlr.press/v97/le19a.html.

H.M.L participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted the experiments (with the help from Voloshin), and participated in the writing of the manuscript.

Hoang M Le, Peter Carr, Yisong Yue, and Patrick Lucey. Data-driven ghosting using deep imitation learning. URL http://www.sloansportsconference.com/wp-content/uploads/2017/02/1671-2.pdf.

H.M.L participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted the experiments, and participated in the writing of the manuscript.

Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. In *International Conference on Machine Learning*, pages 1995–2003, 2017. URL http://proceedings.mlr.press/v70/

le17a.html.
H.M.L participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted the experiments, and participated in the writing of the manuscript.

A. J. Taylor, V. D. Dorobantu, H. M. Le, Y. Yue, and A. D. Ames. Episodic learning with control lyapunov functions for uncertain robotic systems*. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6878–6884, 2019. URL https://doi.org/10.1109/IROS40897.2019.8967820.
H.M.L participated in analyzing the method and writing of the manuscript.

Abhinav Verma, Hoang Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 15726–15737, 2019. URL https://papers.nips.cc/paper/9705-imitation-projected-programmatic-reinforcement-learning.pdf.
H.M.L participated in the conception of the project, developed the algorithm, performed theoretical analysis, and participated in the writing of the manuscript.

Cameron Voloshin, Hoang M Le, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*, 2019. URL https://arxiv.org/abs/1911.06854.
H.M.L participated in the conception of the project, analyzed the data, and participated in the writing of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Structured Policy Learning - Definition and Classification

A Real-World Decision Making Motivation. The goal of this thesis is to make some contributions towards improving learning-based sequential decision making systems that are broadly beneficial for real-world applications. For the purpose of this thesis, we will consider sequential decision making to be the task of making a series of decision in a dynamic environment overtime, to achieve certain specified outcome. While the specific objective of decision making may vary depending on application domain, it is not difficult to appreciate the scope of dynamic decision making systems. Traffic routing, energy and power systems, industrial robots, warehouse logistics, are among long-standing examples of sequential decision making systems that have had large impact on daily lives. Clearly, not all previous examples of decision making systems are what we would currently consider fully algorithmic, or learning-based systems. However, from conversational agents to online recommendation to search and advertising, we are clearly already interacting with increasingly sophisticated learning-based systems on a daily basis. The trends toward more data-driven and algorthmic-driven systems are rooted in exciting developments in machine learning, aided by improving computational power and increasing availability of data. Supervised machine learning has made significant advances in the past decade, with highly visible successes in the areas of computer vision (Krizhevsky et al., 2012; Deng et al., 2009) and natural language processing (Devlin et al., 2019). Powerful deep learning advances have also contributed to recent advances in sequential decision making domains, perhaps most notably computer games (Mnih et al., 2015), and board games (Silver et al., 2017), and robotic manipulations (Andrychowicz et al., 2020). Looking ahead into the near future, researchers and practitioners are exploring potential applications in other areas such as healthcare, remote work and education, autonomous vehicles, automated customer and decision support, etc. Two central questions for these aspirational applications natually arise: *(i) how do we approach these diverse problem areas from a machine learning perspective? and (ii) how do we use learning to build real-world systems in a way that is computationally efficient, safe, and grounded in real-world constraints?*

Policy Learning in Dynamic Environment. This dissertation focuses on the problem of policy learning for sequential decision making tasks. A policy encapsulates the decision making mechanism of an agent interacting with its environment over time. At a high-level, this interaction can be viewed via an abstraction similar to the standard reinforcement learning framework (Sutton and Barto, 2018b; Bertsekas et al., 2005). The protocol is such that at each time step, the environment reveals a context, the agent takes an action given the context, and the action influences the context that the agent observes in the following time step, and so forth. Notice that in general, this abstraction can be more general than a Markov Decision Process (Puterman, 2014). Nevertheless, a policy is simply a mapping from context to (a probability distribution over) actions. Often, the policy learning goal is to derive an optimal policy, where optimality criterion depends on the exact problem formulation.

We will give a more in-depth account of contemporary approaches to policy learning in the next chapter. As a rough summary, many recent work in policy learning focus on balancing the exploration-exploitation trade-off, or casting the interaction paradigm under reinforcement / imitation learning dichotomy. When the dynamics are unknown, **reinforcement learning** (RL) is a general class of samples-based techniques to learn policies, where samples often come from exploring the environment Sutton and Barto (2018b). Efficient exploration is a major challenge of reinforcement learning, and is the subject of very active research (Jiang et al., 2017). Even for multi-arm bandit problem (RL with horizon 1), learning an ϵ-optimal policy may require $O(\frac{1}{\epsilon^2})$ samples in the worst case (Krishnamurthy et al., 2016). One alternative class of techniques that short-circuit the exploration issue is **imitation learning** (IL). Imitation learning is a direct way to learn policy by from expert oracle, which can be computational oracle, or human expert. Thus imitation learning can be viewed as generalization of supervised learning to the sequential decision domains.

Some Challenges with Contemporary Approaches. Both reinforcement and imitation learning are general purpose classes of methods. However, the success stories of both groups of techniques, while impressive (Cheng et al., 2019c; Pan et al., 2018; Finn et al., 2016; Mnih et al., 2016; Silver et al., 2017), are still constrained to somewhat limited domains, where we either have abundant access to simulated data, or strong assumptions on the oracles.

Even with increasing computational power, *data efficiency* is still paramount. For

RL, much recent research effort has been dedicated towards improving sample efficiency of exploration (Kakade et al., 2003; Agarwal et al., 2014; Jiang et al., 2017). Imitation learning generally has been shown to be more sample-efficient than pure exploration-based RL (Sun et al., 2017; Ross and Bagnell, 2014). However, the sample cost to the expert oracle can potentially much more expensive than the sample cost for exploration in scenarios where simulation-to-real transfer is feasible (Peng et al., 2018), or in video games domains where simulation matches reality.

The learning agent has to handle *imperfect cost and observation*. Engineering the right cost (reward) function is difficult for many realistic scenarios. In addition, the most common modeling paradigm in RL is via Markov Decision Processess, which assume full observation (context) of the learning agent. In practice, partial observability is a prevalent setting and policy learning has to account for imperfect information.

Finally, we would like to emphasize additional *real-world constraints* introduced by various domains that are not adequately addressed by classical learning techniques. For policy learning methods to be trusted with realistic applications, they need to meet certain domain-specific desiderata:

- One sensible requirement is safety, concerning much of the applications related to human-computer interaction, such as autonomous driving, healthcare, education, etc. (Garcıa and Fernández, 2015) At the most basic level, random exploration is orthogonal to safe learning, thus posing challenges to purely exploration-based RL
- Other reasonable constraints may include: *stability*, which is among the most important concerns for roboticists and control engineers (Berkenkamp et al., 2017); *verifiability* (or provably correctness in the formal methods sense); *smoothness* requirement for many human-facing decision making tasks, such as driving; (Bojarski et al., 2016); *fairness* of policy learning in the context of socio-technical systems (Liu et al., 2018a).

Im summary, despite exciting developments, there remains a sizable gap between what current learning-based methods could achieve and broadly useful, real-world applications of policy learning across different domains.

Structured Policy Learning. This dissertation describes recent advances that attempt to narrow the gap between contemporary policy learning methods and

realistic sequential decision making domains. The proposed methods center around the theme of **structured policy learning**, which we define as the class of techniques that integrates *principled use of domain knowledge* into the policy learning process, in order to satisfy real-world desiderata as described above. A natural question given this definition is how should we further clarify the source of *domain knowledge*? While it is tempting to cast a stereotypical RL technique as assuming "no structure", and a stereotypical IL technique as assuming a strong form of structure, we argue that in the context of realistic constraints, this distinction is not always the most helpful. Consider two examples:

- Consider a hypothetical RL scenario with idealized cost function, where single-stage cost (reward) matches exactly to the consecutive difference (differential) of the optimal reward function. In this setting, optimal cost (reward) design plays the role of the expert Ross and Bagnell (2014) and alleviates the difficulty that is associated with a long horizon problem with sparse cost (reward) signals. While this is an idealized scenario, we can view leveraging off-line (sub-optimal) data, or value specification associated with realistic requirements such as safety, are among possible venues to encode structure into the learning problem.

- Consider a hypothetical IL scenario where a computational expert is easy to derive via prior models (e.g., physics-based optimal controller for robotics, or rule-based experts from existing systems). In this setting, perhaps standard imitation learning techniques could succeed in learning the expert's policy. However, in case of an imperfect model, the cost (reward) corresponding to the expert could be suboptimal in reality. The question facing the learning agent is then how to reconcile and possibly leverage the information coming from the expert in order to move towards another optimality criterion?

In examining these two examples above, we can see that integrating the right amount of domain knowledge is not necessarily about forcing a trade-off between RL and IL. In fact, as we will show in this dissertation, in many cases leveraging structure can be a complementary abstraction to improve the learning performance while treating the underlying policy learning sub-routine as a flexible reduction.

In this thesis, we will discuss three different ways to integrate structure into policy learning:

1. **Value-based structure:** the desired objective can be expressed via some constraints on the value function of a policy, where the value comes from prior knowledge (e.g., safety), or historical data (e.g., policy improvement).

2. **Policy-based structure:** the cost (reward) might be difficult to specify, and it can be easier to build the structure more directly into the policy class (e.g., policy is locally stable)

3. **Model-based structure:** An (approximate) knowledge about the environment or the task is attainable (e.g., known hierarchical structure, or approximate physics-based dynamics model)

We aim to incorporate these structures into the design of learning algorithms and show how they can help improve both data efficiency and constraint satisfactions. We will discuss methodological advances as well as empirical validations. For the remainder of this chapter, we briefly introduce several concrete research directions and the corresponding contributions of this dissertation.

1.2 Learning with Value-Based Structure

This is a setting where the domain expert can naturally specify constraints on value function, or cost function objectives. In chapter 2, we consider a reinforcement learning scenario where different value-based constraints can be imposed on a policy learning problem in a counterfactual manner, i.e., batch setting. The off-line, off-policy learning approach is promising for realistic applications where existing data is available, but the standard cost objective is not sufficient for learning a reliable decision making agent. We propose a meta learning algorithm, and instantiate with specific subroutines that offer end-to-end finite-sample performance guarantees. The finite-sample results link the number of off-policy samples with how well the algorithm can return an optimal policy subject to satisfying the value-based constraints. To certify constraint satisfaction, we propose a new off-policy policy evaluation (OPE) algorithm that is applicable in other reinforcement learning settings.

Reasoning about value-based structure in a data-efficienct way requires off-policy evaluation of policy performance as a natural component. Off-policy reasoning is highly relevant for any safe and efficient deployment of policy learning to realistic domains. In Chapter 3, we take a detailed look into the empirical performance of different off-policy estimators. We design experimental conditions that highlight

different factors that are often neglected in previous work on OPE. Through our analysis, we contribute a summary guideline for method selection. To the best of our knowledge, Chapter 3 is the first comprehensive study of contemporary OPE methods.

1.3 Learning with Policy-Based Structure

When the value-based evaluation of the constraints are hard to determined, it can be more natural to build the constraints directly into the policy class. The learning policy can then inherit the structure of the policy class, i.e., the desired structure can be (approximately) "correct-by-construction", which is an appealing property. For policy-based structure, the challenge lies in designing algorithms that permit reasoning about other optimality criteria of learning.

Similar to how regularization encodes prior knowledge in supervised learning context, we take a policy class regularization perspective to policy learning in the sequential setting. In this case, regularizing with respect to a policy class allows mediating between an expressive policy class, such as deep neural networks, and less expressive policy class, but with certifiable properties.

In Chapter 4, we regularize to a smooth policy class (under precise notion of smoothness) and designed a learning algorithm that takes a reduction approach to imitation learning. We prove that exploiting this policy structure improve data efficiency of learning. The learning algorithm also outputs learned policy that has better smoothness property than using conventional imitation learning alone. We demonstrate the empirical performance of the algorithm in an realistic applications that require learning smooth decision making policies.

In Chapter 5, we generalize this approach to the RL setting where we leverage reduction to online learning, in particular mirror descent, in the policy space. Similar to Chapter 4, regularization to a policy class is enforced via projecting onto a constrained policy space. The projection of the online learning reduction is equivalent to an IL subroutine, where the learning policy learns from the best "expert" from the prescribed policy structure. The update step of online learning can assume the form of standard RL update. In addition to the convergence analysis, in this chapter we demonstrate a potential application in learning "programmatic" policies using RL, where the programs are representable using structured programming language. The algorithm aims to output a near optimal programmatic policy using this lift-and-project approach. We show in a driving simulation that the learned policy can

generalize better to unseen environment compared to policies learned purely from RL procedures. In addition, programmatic policies are also amenable to formal verification, whereas deep neural networks remain difficult to verify using current techniques.

1.4 Learning with Model-Based Structure

Distinct from model-based RL, the model-based structure refers to partial knowledge of the environment or the task, which the learning agent can exploit to help speed up learning process. Below are some examples of model-based structure.

Hierarchical structure is well studied in machine learning. In fact, recent success of computer vision can be attributed to good hierarchical feature learning LeCun et al. (1995); Qi et al. (2017). In reinforcement learning context, a common strategy to improve sample efficiency in RL over long time horizons is to exploit hierarchical structure of the problem. In Chapter 6, we propose methods to improve the sample-efficiency of both imitation learning and reinforcement learning, taking into account the hierarchy decomposition. Our key design principle is an algorithmic framework called *hierarchical guidance*, in which feedback (labels) from the high-level expert is used to focus (guide) the low-level learner. The high-level expert ensures that low-level learning only occurs when necessary (when subtasks have not been mastered) and only over relevant parts of the state space. This differs from a naïve hierarchical approach which merely gives a subtask decomposition. Focusing on relevant parts of the state space speeds up learning (improves sample efficiency), while omitting feedback on the already mastered subtasks reduces expert effort (improves label efficiency). We validate the approach in long horizon domains, including the challenging Atari game Montezuma Revenge. Our experiments show that incorporating a modest amount of expert feedback can lead to dramatic improvements in performance compared to pure hierarchical RL.

Latent structure Reinforcement learning under partial observability is intractable in general POMDP (Papadimitriou and Tsitsiklis, 1987). Recent progress has been made towards policy learning in contextual decision processes under low rank structural assumptions (Jiang et al., 2017). The difficulty of policy learning is compounded for multi-agent settings, where information regarding the intention of other agents in the environment is missing. In Chapter 7, we study the problem of policy learning for multiple coordinating agents from demonstrations. Many realistic multi-agent settings require coordination among collaborative agents to

achieve some common goal: modeling team sports, learning policies for game AI, controlling teams of multiple robots, or modeling collective animal behavior. Typically, the agents have access to the outcome of actions from other agents, but the coordination mechanism is neither clearly defined nor observed, which makes the full state only partially observable.

We propose a semi-supervised learning framework that integrates and builds upon conventional imitation learning and unsupervised structure learning. The latent structure is represented by a graphical model, which encodes a coordination mechanism of interacting agents. In order to make learning tractable, we develop an alternating optimization method that enables integrated and efficient training of both individual policies and the latent structure model. For learning individual policies, we extend reduction-based single-agent imitation learning approaches into multi-agent domain, utilizing powerful black-box supervised techniques such as deep learning as base routines. For latent structure learning, we develop a stochastic variational inference approach. We show that learning a good latent structure to encode implicit coordination yields significantly superior imitation performance compared to conventional baselines.

1.5 Overview Discussion on Existing Approaches to Policy Learning

A sequential decision making agent is characterized by its interaction with the environment (Sutton and Barto, 2018b; Bertsekas et al., 2005). For each time t, the environment reveals context x_t in the context space X, the agent takes an action $a_t \in \mathbf{A}$, and the action influences the context x_{t+1} that the agent observes in the following time step, and so forth. This interaction paradigm is general, in the sense

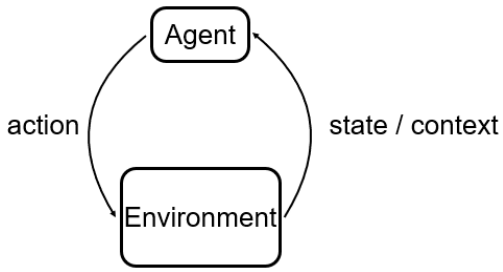


Figure 1.1: Policy maps context to actions

that the context can capture partial observation of the true state, and the agent-environment interaction can also stand for an interaction between an agent and the expert oracle. As such, this abstraction can convey both the reinforcement learning

and imitation learning paradigms. A policy π is simply a mapping from context $x \in \mathbf{X}$ to (a probability distribution over) actions in A. As mentioned in the previous chapter, the optimization objective of policy learning depends on the problem setup.

In learning-based approaches, the dynamical system characterizing the agent - environment interaction is typically viewed as unknown or uncertain. If the dynamics are known, one can leverage well-established techniques in dynamic programming and optimal control to solve for optimal policies (Bertsekas et al., 2005). When the dynamics are unknown, reinforcement learning is a general class of sample-based methods to solve optimal control problems (Sutton and Barto, 2018b). Consider a simple and stereotypical example of sequential decision making: a navigation task over a grid where an agent has to make a series of steps towards the goal while avoiding obstacles along the way. Ideally we would want such a learning agent to be able to generalize, i.e., the configuration of the environment and goal locations might change from episode to episode. Without prior knowledge of environmental dynamics, a reinforcement learning agent needs to explore the environment to search for an optimal policy. The objective in RL is usually to minimize long-term cost (or equivalently, maximizing rewards): $C(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(x_t, a_t)\right]$, where $C(\pi)$ is called the value function of policy π, and $c(x_t, a_t)$ denotes single-stage cost (reward) associated with observation x_t and action a_t. In the last few years, the combination of deep learning and classical RL techniques have produced very impressive empirical successes mentioned above: from playing Atari games (Mnih et al., 2015), to playing board games (Silver et al., 2017), to certain robotic manipulation tasks (Andrychowicz et al., 2020). Without knowing the dynamics, a learning agent needs to interact with the environment for exploration, before a good policy can be learned via optimization (exploitation). Consequently, success in reinforcement learning is still dependent on having access to environments where simulation data is cheap, as the number of interaction with the environment is significant. Bringing reinforcement learning techniques to real-world applications may introduce several challenges:

1. *Well-defined cost objective*: The underlying assumption in RL framework is that what the agent optimizes for can be summarized in a well-defined, scalar cost function. Even if such hypothesis is valid, one can imagine designing a good cost function is difficult in many scenarios (e.g., self-driving vehicles). The difficulty of designing cost (reward) function motivates many recent effort to investigate this inverse problem Hadfield-Menell et al. (2017); Finn et al.

(2016); Ziebart et al. (2008); Sorg et al. (2010). Designing an optimal cost function can be as hard as solving the forward problem in the worst case Ng et al. (1999).

2. *Exploration*: Even when we can obtain a reasonable cost function and observe full state, the learning agent may come up against the third issue, which is how to explore the environment efficiently. Efficient exploration is the holy-grail of reinforcement learning, and is the subject of very active research (Jiang et al., 2017). Even for multi-arm bandit problem (RL with horizon 1), learning an ϵ-optimal policy may require $O(\frac{1}{\epsilon^2})$ samples in the worst case (Krishnamurthy et al., 2016). One side effect of random exploration is that the policy may exhibit undesirable properties during training. Given the difficulty of exploration in general, some recent work have developed more efficient exploration algorithms under structural assumptions. (Du et al., 2019) proved an efficient algorithm is possible assuming latent block structure of underlying contextual space. (Misra et al., 2019) built on similar theme and showed that exploiting state abstraction can make hard exploration problems tractable.

One alternative class of techniques that short-circuit the exploration issue is imitation learning (IL). Imitation learning is a direct way to learn policy by leveraging domain knowledge. Often the domain knowledge comes from optimal oracle, which can be computational oracle, or human expert. The goal in imitation learning is usually minimize the imitation loss with respect to the expert policy, in that sense the cost function is easy to define: $C(\pi) = \mathbb{E}_{d_\pi}[\ell(\pi(x_t), \pi^*(x_t))]$, where π^* is the expert policy, and the expectation is taken over state distribution d_π induced by the learning policy π.

Many imitation learning techniques involve reductions to supervised learning. Going back to the navigation example, we can obtain a data set from querying oracle policy π^*, which is either a search or optimization oracle, or human expert to get the shortest path. We can simply reduce the sequential decision problem to supervised learning, e.g., fitting a neural network to directly find the mapping from X to A in this data set. This simple strategy is also known as behavioral cloning, which was used for one of the first autonomous driving system, ALVINN (Pomerleau, 1989).

Behavioral cloning is simple and straightforward to implement, and may in fact work well when data coverage is sufficient. However in many cases, especially under partial observability, a behaviorally cloned policy may fail to generalize to

unseen instances of the environment. Interactive imitation learning is an alternative strategy to improve upon this weakness. Specifically, learning proceeds over multiple episodes, similar to many episodic reinforcement learning techniques. At each round of learning, the current policy, which might be performing poorly, can query online expert to yield the correct action labels corresponding to the executed trajectory. With each episode, querying the interactive oracle yields a new data set, which can again be used in some update supervised learning procedure. As such, we reduce the sequential decision problem to multiple supervised learning problems. To update the learning policy, we can use either policy aggregation (Daumé III et al., 2009), or data aggregation, in the style known as DAgger (Ross et al., 2011b). These are among the most common imitation learning techniques that have found widespread successes.

Overall, imitation learning can be significantly more sample efficient than pure exploration-based RL Sun et al. (2017). Notice that the two classes of techniques can be complementary: several recent RL success stories rely on bootstrapping the learning agent with a large repository of expert trajectories to increase data and computational efficiency (Silver et al., 2016; Vinyals et al., 2019). Many recent successes from imitation learning are likely unattainable from conventional RL techniques: autonomous driving of a real race car (Pan et al., 2018), visual speech animation (Taylor et al., 2017), or more data-efficient robotic manipulation (Duan et al., 2017). These are examples of tasks that are very difficult to do with deep RL techniques alone, either due to a lack of natural simulation environment, or unacceptable sample efficiency, or both. Still, there are two main drawbacks of imitation learning strategy:

1. *Hard to obtain expert labels*: Imitation learning can be viewed as generalization of supervised learning to the sequential decision domains. Similar to supervised learning, high-quality labels are required for good learning performance. In many applications, it can be expensive to obtain expert oracle's labels, especially when interactive expert is part of the learning loop. For computational expert, computational time can be an acceptable trade-off for data acquisition. However, this issue is notable when the agent is learning with human expert in the loop. Imitation learning techniques generally should address the problem of data efficiency for the expert.

2. *Non-optimal expert*: Second, expert oracle itself may not be optimal, or fully consistent from one expert to another. If the expert policy is not optimal, then

the cost objective to learn the policy is also faulty. Here the question of cost function design is similar to the challenge we face in reinforcement learning case.

In this section, we give a background overview for conventional approaches to policy learning. In the next chapters, as we develop our approaches to *structured policy learning*, we will further provide details from relevant recent work that combines policy learning with different forms of structural considerations.

Chapter 2

POLICY LEARNING UNDER VALUE-BASED CONSTRAINTS

Summary. When learning policies for real-world domains, two important questions arise: (i) how to efficiently use pre-collected off-policy, non-optimal behavior data; and (ii) how to mediate among different competing objectives and constraints. We thus study the problem of batch policy learning under multiple constraints, and offer a systematic solution. We first propose a flexible meta-algorithm that admits any batch reinforcement learning and online learning procedure as subroutines. We then present a specific algorithmic instantiation and provide performance guarantees for the main objective and all constraints. As part of off-policy learning, we propose a simple method for off-policy policy evaluation (OPE) and derive PAC-style bounds. Our algorithm achieves strong empirical results in different domains, including in a challenging problem of simulated car driving subject to multiple constraints such as lane keeping and smooth driving. We also show experimentally that our OPE method outperforms other popular OPE techniques on a standalone basis, especially in a high-dimensional setting.

2.1 Policy Learning under Value-Based Constraints

We study the problem of policy learning under multiple constraints. Contemporary approaches to learning sequential decision making policies have largely focused on optimizing some cost objective that is easily reducible to a scalar value function. However, in many real-world domains, choosing the right cost function to optimize is often not a straightforward task. Frequently, the agent designer faces multiple competing objectives. For instance, consider the aspirational task of designing autonomous vehicle controllers: one may care about minimizing the travel time while making sure the driving behavior is safe, consistent, or fuel efficient. Indeed, many such real-world applications require the primary objective function be augmented with an appropriate set of constraints (Altman, 1999).

Contemporary policy learning research has largely focused on either online reinforcement learning (RL) with a focus on exploration, or imitation learning (IL) with a focus on learning from expert demonstrations. However, many real-world settings already contain large amounts of pre-collected data generated by existing policies (e.g., existing driving behavior, power grid control policies, etc.). We thus study the

complementary question: *can we leverage this abundant source of (non-optimal) behavior data in order to learn sequential decision making policies with provable guarantees on both primary objective and constraint satisfaction*?

We thus propose and study the problem of batch policy learning under multiple constraints. Historically, batch RL is regarded as a subfield of approximate dynamic programming (ADP) (Lange et al., 2012), where a set of transitions sampled from the existing system is given and fixed. From an interaction perspective, one can view many online RL methods (e.g., DDPG (Lillicrap et al., 2016)) as running a growing batch RL subroutine per round of online RL. In that sense, *batch policy learning is complementary to any exploration scheme.* To the best of our knowledge, the study of constrained policy learning in the batch setting is novel.

We present an algorithmic framework for learning sequential decision making policies from off-policy data. We employ multiple learning reductions to online and supervised learning, and present an analysis that relates performance in the reduced procedures to the overall performance with respect to both the primary objective and constraint satisfaction.

Constrained optimization is a well studied problem in supervised machine learning and optimization. In fact, our approach is inspired by the work of (Agarwal et al., 2018) in the context of fair classification. In contrast to supervised learning for classification, batch policy learning for sequential decision making introduces multiple additional challenges. First, setting aside the constraints, batch policy learning itself presents a layer of difficulty, and the analysis is significantly more complicated. Second, verifying whether the constraints are satisfied is no longer as straightforward as passing the training data through the learned classifier. In sequential decision making, certifying constraint satisfaction amounts to an off-policy policy evaluation problem, which is a challenging problem and the subject of active research. In this chapter, we develop a systematic approach to address these challenges, provide a careful error analysis, and experimentally validate our proposed algorithms. In summary, our contributions in this chapter are:

- We formulate the problem of batch policy learning under multiple constraints, and present the first approach of its kind to solve this problem. The definition of constraints is general and can subsume many objectives. Our approach utilizes multi-level learning reductions, and we show how to instantiate it using various batch RL and online learning subroutines. We show that guarantees from the subroutines provably lift to provide end-to-end guarantees on the

original constrained batch policy learning problem.

- While leveraging techniques from batch RL as a subroutine, we provide a refined theoretical analysis for general non-linear function approximation that improves upon the previously known sample complexity result (Munos and Szepesvári, 2008).
- To evaluate off-policy learning performance and constraint satisfaction, we propose a simple new technique for off-policy policy evaluation (OPE), which is used as a subroutine in our main algorithm. We show that it is competitive to other OPE methods.
- We validate our algorithm and analysis with two experimental settings. First, a simple navigation domain where we consider safety constraint. Second, we consider a high-dimensional racing car domain with smooth driving and lane centering constraints.

2.2 Problem Formulation in the Batch Setting

We first introduce notation. Let $\mathbf{X} \subset \mathbb{R}^d$ be a bounded and closed d-dimensional state space. Let $\mathbf{A}$ be a finite action space. Let $c : \mathbf{X} \times \mathbf{A} \mapsto [0, \bar{C}]$ be the primary objective cost function that is bounded by $\bar{C}$. Let there be m constraint cost functions, $g_i : \mathbf{X} \times \mathbf{A} \mapsto [0, \bar{G}]$, each bounded by $\bar{G}$. To simplify the notation, we view the set of constraints as a vector function $g : \mathbf{X} \times \mathbf{A} \mapsto [0, \bar{G}]^m$ where $g(x, a)$ is the column vector of individual $g_i(x, a)$. Let $p(\cdot|x, a)$ denote the (unknown) transition/dynamics model that maps state/action pairs to a distribution over the next state. Let $\gamma \in (0, 1)$ denote the discount factor. Let χ be the initial states distribution.

We consider the discounted infinite horizon setting. An MDP is defined using the tuple $(\mathbf{X}, \mathbf{A}, c, g, p, \gamma, \chi)$. A policy $\pi \in \Pi$ maps states to actions, i.e., $\pi(x) \in \mathbf{A}$. The value function $C^\pi : \mathbf{X} \mapsto \mathbb{R}$ corresponding to the primary cost function c is defined in the usual way: $C^\pi(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(x_t, a_t) \mid x_0 = x\right]$, over the randomness of the policy π and transition dynamics p. We similarly define the vector-value function for the constraint costs $G^\pi : \mathbf{X} \mapsto \mathbb{R}^m$ as $G^\pi(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t g(x_t, a_t) | x_0 = x\right]$. Define $C(\pi)$ and $G(\pi)$ as the expectation of $C^\pi(x)$ and $G^\pi(x)$, respectively, over the distribution χ of initial states.

Batch Policy Learning under Constraints

In batch policy learning, we have a pre-collected dataset:
$\mathrm{D} = \{(x_i, a_i, x_i', c(x_i, a_i), g_{1:m}(x_i, a_i)\}_{i=1}^n$, generated from (a set of) historical behavioral policies denoted jointly by π_D. The goal of batch policy learning under constraints is to learn a policy $\pi \in \Pi$ from D that minimizes the primary objective cost while satisfying m different constraints:

$$\begin{aligned} \min_{\pi \in \Pi} \quad & C(\pi) \\ \text{s.t.} \quad & G(\pi) \leqslant \tau \end{aligned} \tag{OPT}$$

where $G(\cdot) = [g_1(\cdot), \ldots, g_m(\cdot)]^\top$ and $\tau \in \mathbb{R}^m$ is a vector of known constants. We assume that (OPT) is feasible. However, the dataset D might be generated from multiple policies that violate the constraints.

Examples of Policy Learning with Constraints

Counterfactual & Safe Policy Learning. In conventional online RL, the agent needs to "re-learn" from scratch when the cost function is modified. Our framework enables counterfactual policy learning assuming the ability to compute the new cost objective from the same historical data. A simple example is *safe* policy learning (Garcıa and Fernández, 2015). Define safety cost $g(x, a) = \phi(x, a, c)$ as a new function of existing cost c and features associated with current state-action pair. The goal here is to counterfactually avoid undesirable behaviors observed from historical data. We experimentally study this safety problem in Section 2.5.

Other examples from the literature that belong to this safety perspective include planning under chance constraints (Ono et al., 2015; Blackmore et al., 2011). The constraint here is $G(\pi) = \mathbb{E}[\mathbb{I}(x \in \mathbf{X}_{error})] = \mathrm{P}(x \in \mathbf{X}_{error}) \leqslant \tau$.

Multi-objective Batch Learning. Traditional policy learning (RL or IL) presupposes that the agent optimizes a single cost function. In reality, we may want to satisfy multiple objectives that are not easily reducible to a scalar objective function. One example is learning fast driving policies under multiple behavioral constraints such as smooth driving and lane keeping consistency (see Section 2.5).

Equivalence between Constraint Satisfaction and Regularization

Our constrained policy learning framework accommodates several existing regularized policy learning settings. Regularization typically encodes prior knowledge, and has been used extensively in the RL and IL literature to improve learning per-

formance. Many instances of regularized policy learning can be naturally cast into (OPT):

- *Entropy regularized RL* (Haarnoja et al., 2017; Ziebart, 2010) maps to policy-dependent constraint cost $g(x) = \mathbb{H}(\pi(\cdot|x))$, where $\mathbb{H}$ measures conditional entropy.[1]
- *Conservative policy improvement* (Levine and Abbeel, 2014; Schulman et al., 2015; Achiam et al., 2017) is equivalent to $G(\pi) = D_{KL}(\pi, \pi_k)$, where π_k is some "well-behaving" policy.
- *Smooth imitation learning* (Le et al., 2016a) is equivalent to $G(\pi) = \min_{h \in \mathrm{H}} \Delta(h, \pi)$, where H is a class of provably smooth policies and Δ is a divergence metric.
- *Regularizing RL with expert demonstration* (Hester et al., 2018) is equivalent to $G(\pi) = \mathbb{E}[\ell(\pi(x), \pi^*(x))]$, where π^* is the expert policy.

We provide further equivalence derivation of the above examples in Appendix A.1. Of course, some problems are more naturally described using the regularization perspective, while others using constraint satisfaction.

More generally, one can establish the equivalence between regularized and constrained policy learning via a simple appeal to Lagrangian duality as shown in Proposition 2.2.1 below. This Lagrangian duality also has a game-theoretic interpretation (Section 5.4 of (Boyd and Vandenberghe, 2004)), which serves as an inspiration for developing our approach.

Proposition 2.2.1. *Let Π be a convex set of policies. Let $C : \Pi \mapsto \mathbb{R}, C : \Pi \mapsto \mathbb{R}^K$ be value functions. Consider the two policy optimization tasks:*

$$\texttt{Regularization:} \quad \min_{\pi \in \Pi} \quad C(\pi) + \lambda^\top G(\pi)$$

$$\texttt{Constraint:} \quad \min_{\pi \in \Pi} \quad C(\pi) \quad \textit{s.t. } G(\pi) \leqslant \tau$$

Assume that the Slater's condition is satisfied in the `Constraint` *problem (i.e., $\exists \pi$ s.t. $G(\pi) < \tau$). Assume also that the constraint cannot be removed without changing the optimal solution, i.e., $\inf_{\pi \in \Pi} C(\pi) < \inf_{\pi \in \Pi : G(\pi) \leqslant \tau} C(\pi)$. Then $\forall\, \lambda > 0$, $\exists\, \tau$, and vice versa, such that* `Regularization` *and* `Constraint` *share the same optimal solutions. (Proof in Appendix A.1.)*

[1] Constraint value function $G(\pi)$ can be viewed as the expectation over discounted state visitation distribution. The lack of explicit discount rate does not intefere with our overall approach.

2.3 Algorithms for Batch Policy Learning under Value-Based Constraints

To make use of strong duality, we first *convexify* the policy class Π by allowing stochastic combinations of policies, which effectively expands Π into its convex hull $\texttt{Conv}(\Pi)$. Formally, $\texttt{Conv}(\Pi)$ contains *randomized policies*, which we denote $\pi = \sum_{t=1}^{T} \alpha_t \pi_t$ for $\pi_t \in \Pi$ and $\sum_{t=1}^{T} \alpha_t = 1$. Executing a mixed π consists of first sampling *one* policy π_t from $\pi_{1:T}$ according to distribution $\alpha_{1:T}$, and then executing π_t. Note that we still have $\mathbb{E}[\pi] = \sum_{t=1}^{T} \alpha_t \mathbb{E}[\pi_t]$ for any first-moment statistic of interest (e.g., state distribution, expected cost). It is easy to see that the augmented version of (OPT) over $\texttt{Conv}(\Pi)$ has a solution at least as good as the original (OPT). As such, to lighten the notation, we will equate Π with its convex hull for the rest of the chapter.

Meta-Algorithm

The Lagrangian of (OPT) is $L(\pi, \lambda) = C(\pi) + \lambda^\top (G(\pi) - \tau)$ for $\lambda \in \mathbb{R}^m_+$. Clearly (OPT) is equivalent to the min-max problem: $\min_{\pi \in \Pi} \max_{\lambda \in \mathbb{R}^k_+} L(\pi, \lambda)$. We assume (OPT) is feasible and that Slater's condition holds (otherwise, we can simply increase the constraint τ by a tiny amount). Slater's condition and policy class convexification ensure that strong duality holds (Boyd and Vandenberghe, 2004), and (OPT) is also equivalent to the max-min problem: $\max_{\lambda \in \mathbb{R}^k_+} \min_{\pi \in \Pi} L(\pi, \lambda)$.

Since $L(\pi, \lambda)$ is linear in both λ and π (due to stochastic mixture[2]) , strong duality is also a consequence of von Neumann's celebrated convex-concave minimax theorem for zero-sum games (Von Neumann and Morgenstern, 2007). From a game-thoeretic perspective, the problem becomes finding the equilibrium of a two-player game between the $\pi-$player and the $\lambda-$player (Freund and Schapire, 1999). In this repeated game, the $\pi-$player minimizes $L(\pi, \lambda)$ given the current λ, and the $\lambda-$player maximizes it given the current (mixture over) π.

We first present a meta-algorithm (Algorithm 1) that uses any no-regret online learning algorithm (for λ) and batch policy optimization (for π). At each iteration, given λ_t, the π-player runs `Best-response` to get the best response:

$$\begin{aligned} \texttt{Best-response}(\lambda_t) &= \arg\min_{\pi \in \Pi} L(\pi, \lambda_t) \\ &= \arg\min_{\pi \in \Pi} C(\pi) + \lambda_t^\top (G(\pi) - \tau). \end{aligned}$$

[2]This places no restrictions on the individual policies. Individual policy can be non-linear and cost function can be non-convex.

Algorithm 1 Meta-algo for Batch Constrained Learning

1: **for** each round t **do**
2: $\pi_t \leftarrow$ `Best-response`(λ_t)
3: $\widehat{\pi}_t \leftarrow \frac{1}{t}\sum_{t'=1}^{t} \pi_{t'}$, $\widehat{\lambda}_t \leftarrow \frac{1}{t}\sum_{t'=1}^{t} \lambda_{t'}$
4: $\mathrm{L}_{\max} = \max_{\lambda} L(\widehat{\pi}_t, \lambda)$
5: $\mathrm{L}_{\min} = L($`Best-response`$(\widehat{\lambda}_t), \widehat{\lambda}_t)$
6: **if** $\mathrm{L}_{\max} - \mathrm{L}_{\min} \leqslant \omega$ **then**
7: Return $\widehat{\pi}_t$
8: **end if**
9: $\lambda_{t+1} \leftarrow$ `Online-algorithm`$(\pi_1, \ldots, \pi_{t-1}, \pi_t)$
10: **end for**

This is equivalent to a standard batch reinforcement learning problem where we learn a policy that is optimal with respect to $c + \lambda_t^\top g$. The corresponding mixed strategy is the uniform distribution over all previous π_t. In response to the $\pi-$player, the $\lambda-$player employs `Online-algorithm`, which can be *any* no-regret algorithm that satisfies:

$$\sum_t L(\pi_t, \lambda_t) \geqslant \max_{\lambda} \sum_t L(\pi_t, \lambda) - o(T)$$

Finally, the algorithm terminates when the estimated primal-dual gap is below a threshold ω (Lines 7-8).

Leaving aside (for the moment) issues of generalization, Algorithm 1 is guaranteed to converge assuming: (i) `Best-response` gives the best single policy in the class, and (ii) $\mathrm{L}_{\max}$ and $\mathrm{L}_{\min}$ can be evaluated exactly.

Proposition 2.3.1. *Assuming (i) and (ii) above, Algorithm 1 is guaranteed to stop and the convergence depends on the regret of* `Online-algorithm`*. (Proof in Appendix A.2.)*

Specific Instantiation of Meta-Algorithm

We now focus on a specific instantiation of Algorithm 1. Algorithm 2 is our main algorithm in this chapter.

Policy Learning. We instantiate `Best-response` with Fitted Q Iteration (FQI), a model-free off-policy learning approach (Ernst et al., 2005). FQI relies on a series of reductions to supervised learning. The key idea is to approximate the true action-value function Q^* by a sequence $\{Q_k \in \mathcal{F}\}_{k=0}^{K}$, where $\mathcal{F}$ is a chosen function class.

Algorithm 2 Constrained Batch Policy Learning

Input: Dataset $\mathrm{D} = \{x_i, a_i, x_i', c_i, g_i\}_{i=1}^n \sim \pi_{\mathrm{D}}$. Online algorithm parameters: ℓ_1 norm bound B, learning rate η

1: Initialize $\lambda_1 = (\frac{B}{m+1}, \ldots, \frac{B}{m+1}) \in \mathbb{R}^{m+1}$
2: **for** each round t **do**
3: Learn $\pi_t \leftarrow \texttt{FQI}(c + \lambda_t^\top g)$ *// FQI with cost $c + \lambda_t^\top g$*
4: Evaluate $\widehat{C}(\pi_t) \leftarrow \texttt{FQE}(\pi_t, c)$ *// Algo 3 with π_t, cost c*
5: Evaluate $\widehat{G}(\pi_t) \leftarrow \texttt{FQE}(\pi_t, g)$ *// Algo 3 with π_t,* cost g
6: $\widehat{\pi}_t \leftarrow \frac{1}{t}\sum_{t'=1}^t \pi_{t'}$
7: $\widehat{C}(\widehat{\pi}_t) \leftarrow \frac{1}{t}\sum_{t'=1}^t \widehat{C}(\pi_{t'}), \widehat{G}(\widehat{\pi}_t) \leftarrow \frac{1}{t}\sum_{t'=1}^t \widehat{G}(\pi_{t'})$
8: $\widehat{\lambda}_t \leftarrow \frac{1}{t}\sum_{t'=1}^t \lambda_{t'}$
9: Learn $\widetilde{\pi} \leftarrow \texttt{FQI}(c + \widehat{\lambda}_t^\top g)$ *// FQI with cost $c + \widehat{\lambda}_t^\top g$*
10: Evaluate $\widehat{C}(\widetilde{\pi}) \leftarrow \texttt{FQE}(\widetilde{\pi}, c), \widehat{G}(\widetilde{\pi}) \leftarrow \texttt{FQE}(\widetilde{\pi}, g)$
11: $\widehat{\mathrm{L}}_{\max} = \max_{\lambda, \|\lambda\|_1 = B} \left(\widehat{C}(\widehat{\pi}_t) + \lambda^\top \left[(\widehat{G}(\widehat{\pi}_t) - \tau)^\top, 0 \right]^\top \right)$
12: $\widehat{\mathrm{L}}_{\min} = \widehat{C}(\widetilde{\pi}) + \widehat{\lambda}_t^\top \left[(\widehat{G}(\widetilde{\pi}) - \tau)^\top, 0 \right]^\top$
13: **if** $\widehat{\mathrm{L}}_{\max} - \widehat{\mathrm{L}}_{\min} \leqslant \omega$ **then**
14: Return $\widehat{\pi}_t$
15: **end if**
16: Set $z_t = \left[(\widehat{G}(\pi_t) - \tau)^\top, 0 \right]^\top \in \mathbb{R}^{m+1}$
17: $\lambda_{t+1}[i] = B \frac{\lambda_t[i] e^{-\eta z_t[i]}}{\sum_j \lambda_t[j] e^{-\eta z_t[j]}} \forall i$ *// $\lambda[i]$ the i^{th} coordinate*
18: **end for**

In Lines 3 & 9, $\texttt{FQI}(c + \lambda^\top g)$ is defined as follows. With Q_0 randomly initialized, for each $k = 1, \ldots, K$, we form a new training dataset $\widetilde{\mathrm{D}}_k = \{(x_i, a_i), y_i\}_{i=1}^n$ where:

$$\forall i: \quad y_i = (c_i + \lambda^\top g_i) + \gamma \min_a Q_{k-1}(x_i', a),$$

and $(x_i, a_i, x_i', c_i, g_i) \sim \mathrm{D}$ (original dataset). A supervised regression procedure is called to solve for:

$$Q_k = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n (f(x_i, a_i) - y_i)^2.$$

FQI returns the policy: $\pi_K = \arg\min_a Q_K(\cdot, a)$. FQI has been shown to work well with several empirical domains: spoken dialogue systems (Pietquin et al., 2011), physical robotic soccer (Riedmiller et al., 2009), and cart-pole swing-up (Riedmiller, 2005), and clinical treatment (Prasad et al., 2017). Another possible model-free subroutine is Least-Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003b). One can also consider model-based alternatives (Ormoneit and Sen, 2002).

Algorithm 3 Fitted Q Evaluation: $\texttt{FQE}(\pi, c)$

Input: Dataset $\mathrm{D} = \{x_i, a_i, x_i', c_i\}_{i=1}^n \sim \pi_\mathrm{D}$. Function class $\mathcal{F}$. Policy π to be evaluated

1: Initialize $Q_0 \in \mathcal{F}$ randomly
2: **for** $k = 1, 2, \ldots, K$ **do**
3: Compute target $y_i = c_i + \gamma Q_{k-1}(x_i', \pi(x_i'))\ \ \forall i$
4: Build training set $\widetilde{\mathrm{D}}_k = \{(x_i, a_i), y_i\}_{i=1}^n$
5: Solve a supervised learning problem:
$Q_k = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n (f(x_i, a_i) - y_i)^2$
6: **end for**

Output: $\widehat{C}^\pi(x) = Q_K(x, \pi(x)) \quad \forall x$

Off-policy Policy Evaluation. A crucial difference between constrained policy learning and existing work on constrained supervised learning is the technical challenge of evaluating the objective and constraints. First, estimating $\widehat{L}(\pi, \lambda)$ (Lines 11-12) requires estimating $\widehat{C}(\pi)$ and $\widehat{G}(\pi)$. Second, any gradient-based approach to `Online-learning` requires passing in $\widehat{G}(\pi) - \tau$ as part of gradient estimate (line 15). This problem is known as the off-policy policy evaluation (OPE) problem: we need to evaluate $\widehat{C}(\pi)$ and $\widehat{G}(\pi)$ having only access to data $\mathrm{D} \sim \pi_\mathrm{D}$

There are three main contemporary approaches to OPE: (i) importance weighting (IS) (Precup et al., 2000, 2001), which is unbiased but often has high-variance; (ii) regression-based direct methods (DM), which are typically model-based (Thomas and Brunskill, 2016b),and can be biased but have much lower variance than IS; and (iii) doubly-robust techniques (Jiang and Li, 2016a; Dudík et al., 2011b), which combine IS and DM.

We propose a simple model-free technique using function approximation, called Fitted Q Evaluation (FQE). FQE is based on an iterative reductions scheme similar to FQI, but for the problem of off-policy evaluation. Algorithm 3 lays out the steps. The key difference with FQI is that the *min* operator is replaced by $Q_{k-1}(x_i', \pi(x_i'))$ (Line 3 of Algorithm 3). Each x_i' comes from the original D. Since we know $\pi(x_i')$, each $\widetilde{\mathrm{D}}_k$ is well-defined. Note that FQE can be plugged-in as a direct method if one wishes to augment the policy evaluation with a doubly-robust technique.

Online Learning Subroutine. As $L(\pi_t, \lambda)$ is linear in λ, many online convex optimization approaches can be used for `Online-algorithm`. Perhaps the simpliest choice is Online Gradient Descent (OGD) (Zinkevich, 2003). We include an instantiation using OGD in Appendix A.7.

For our main Algorithm 2, similar to (Agarwal et al., 2018), we use Exponenti-

ated Gradient (EG) (Kivinen and Warmuth, 1997), which has a regret bound of $O(\sqrt{\log(m)T})$ instead of $O(\sqrt{mT})$ as in OGD. One can view EG as a variant of Online Mirror Descent (Nemirovsky and Yudin, 1983) with a softmax link function, or of Follow-the-Regularized-Leader with entropy regularization (Shalev-Shwartz et al., 2012). Gradient-based algorithms generally require bounded λ. We thus force $\|\lambda\|_1 \leqslant B$ using hyperparameter B. Solving (OPT) exactly requires $B = \infty$. We will analyze Algorithm 2 with respect to finite B. With some abuse of notation, we augment λ into a $(m+1)-$dimensional vector by appending $B - \|\lambda\|_1$, and augment the constraint cost vector g by appending 0 (Lines 11, 12 & 15 of Algorithm 2).[3]

2.4 Theoretical Analysis of Proposed Algorithms

Convergence Guarantee

The convergence rate of Algorithm 2 depends on the radius B of the dual variables λ, the maximal constraint value $\bar{G}$, and the number of constraints m. In particular, we can show $O(\frac{B^2}{\omega^2})$ convergence for primal-dual gap ω.

Theorem 2.4.1 (Convergence of Algorithm 2)**.** *After T iterations, the empirical duality gap is bounded by*

$$\widehat{\mathrm{L}}_{\max} - \widehat{\mathrm{L}}_{\min} \leqslant 2\frac{B\log(m+1)}{\eta T} + 2\eta B\bar{G}^2$$

Consequently, to achieve the primal-dual gap of ω, setting $\eta = \frac{\omega}{4\bar{G}^2 B}$ will ensure that Algorithm 2 converges after $\frac{16B^2\bar{G}^2\log(m+1)}{\omega^2}$ iterations. (Proof in Appendix A.2.)

Convergence analysis of our main Algorithm 2 is an extension of the proof to Proposition 2.3.1, leveraging the no-regret property of the EG procedure (Shalev-Shwartz et al., 2012).

Generalization Guarantee of FQE and FQI

In this section, we provide sample complexity analysis for FQE and FQI as *standalone* procedures for off-policy evaluation and off-policy learning. We use the notion of pseudo-dimension as capacity measure of non-linear function class $\mathcal{F}$ (Friedman et al., 2001). Pseudo-dimension $\texttt{dim}_{\mathcal{F}}$, which naturally extends VC dimension into the regression setting, is defined as the VC dimension of the function class induced by the sub-level set of functions of $\mathcal{F}$: $\texttt{dim}_{\mathcal{F}} = \texttt{VC-dim}(\{(x,y) \mapsto$

[3]The $(m+1)^{th}$ coordinate of g is thus always satisfied. This augmentation is only necessary when executing EG.

$\text{sign}(f(x) - y) : f \in \mathcal{F}\})$. Pseudo-dimension is finite for a large class of function approximators. For example, (Bartlett et al., 2017) bounded the pseudo-dimension of piece-wise linear deep neural networks (e.g., with ReLU activations) as $O(WL \log W)$, where W is the number of weights, and L is the number of layers.

Both FQI and FQE rely on reductions to supervised learning to update the value functions. In both cases, the learned policy and evaluation policy induces a different state-action distribution compared to the data generating distribution μ. We use the notion of concentration coefficient for the worst case, proposed by (Munos, 2003), to measure the degree of distribution shift. The following standard assumption from analysis of related ADP algorithms limits the severity of distribution shift over future time steps:

Assumption 1 (Concentrability coefficient of future state-action distribution)**.** *(Munos, 2003, 2007; Munos and Szepesvári, 2008; Antos et al., 2008a,b; Lazaric et al., 2010, 2012; Farahmand et al., 2009; Maillard et al., 2010)*
Let P^π be the operator acting on $f : \boldsymbol{X} \times \boldsymbol{A} \mapsto \mathbb{R}$ s.t.
$(P^\pi f)(x,a) = \int_{\boldsymbol{X}} f(x', \pi(x'))p(dx'|x,a)$. *Given data generating distribution μ, initial state distribution χ, for $m \geqslant 0$ and an arbitrary sequence of stationary policies $\{\pi_m\}_{m \geqslant 1}$ define the concentration coeffient:*

$$\beta_\mu(m) = \sup_{\pi_1,\ldots,\pi_m} \left\| \frac{d(\chi P^{\pi_1} P^{\pi_2} \ldots P^{\pi_m})}{d\mu} \right\|_\infty$$

We assume $\beta_\mu = (1-\gamma)^2 \sum_{m \geqslant 1} m\gamma^{m-1}\beta_\mu(m) < \infty$.

This assumption is valid for a fairly large class of MDPs (Munos, 2007). For instance β_μ is finite for any finite MDP, or any infinite state-space MDP with bounded transition density.[4] Having a finite concentration coefficient is equivalent the top-Lyapunov exponent $\Gamma \leqslant 0$ (Bougerol and Picard, 1992), which means the underlying stochastic system is stable. We show below a simple sufficient condition for Assumption 1 (albeit stronger than necessary).

Example 2.4.1. Consider an MDP such that for any non-stationary distribution ρ, the marginals over states satisfy $\frac{\rho_x(x)}{\mu_x(x)} \leqslant L$ (i.e., transition dynamics are sufficiently

[4]This assumption ensures sufficient data diversity, even when the executing policy is deterministic. An example of how learning can fail without this assumption is based on the "combination lock" MDP (Koenig and Simmons, 1996). In this deterministic MDP example, β_μ can grow arbitrarily large, and we need an exponential number of samples for both FQE and FQI. See Appendix A.4.

stochastic), and $\exists M : \forall x, a : \mu(a|x) > \frac{1}{M}$ (i.e., the behavior policy is sufficiently exploratory). Then $\beta_\mu \leqslant LM$.

Recall that for a given policy π, the Bellman (evaluation) operator is defined as $(\mathbb{T}^\pi Q)(x,a) = r(x,a) + \gamma \int_{\mathbf{X}} Q(x', \pi(x'))p(dx'|x,a)$. In general $\mathbb{T}^\pi f$ may not belong to $\mathcal{F}$ for $f \in \mathcal{F}$. For FQE (and FQI), the main operation in the algorithm is to iteratively project $\mathbb{T}^\pi Q_{k-1}$ back to $\mathcal{F}$ via $Q_k = \arg\min_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|$. The performance of both FQE and FQI thus depend on how well the function class $\mathcal{F}$ approximates the Bellman operator. We measure the ability of function class $\mathcal{F}$ to approximate the Bellman evaluation operator via the worst-case Bellman error:

Definition 2.4.1 (inherent Bellman evaluation error)**.** Given a function class $\mathcal{F}$ and policy π, the *inherent Bellman evaluation error* of $\mathcal{F}$ is defined as $d^\pi_{\mathcal{F}} = \sup_{g\in\mathcal{F}} \inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi g\|_\pi$ where $\|\cdot\|_\pi$ is the ℓ_2 norm weighted by the state-action distribution induced by π.

We are now ready to state the generalization bound for FQE:

Theorem 2.4.2 (Generalization error of FQE)**.** *Under Assumption 1, for $\epsilon > 0$ & $\delta \in (0,1)$, after K iterations of Fitted Q Evaluation (Algorithm 3), for $n = O\big(\frac{\bar{C}^4}{\epsilon^2}(\log\frac{K}{\delta} + \mathtt{dim}_{\mathcal{F}} \log\frac{\bar{C}^2}{\epsilon^2} + \log \mathtt{dim}_{\mathcal{F}})\big)$, we have with probability $1-\delta$:*

$$\big|C(\pi) - \widehat{C}(\pi)\big| \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\big(\sqrt{\beta_\mu}\,\big(2d^\pi_{\mathcal{F}} + \epsilon\big) + \frac{2\gamma^{K/2}\bar{C}}{(1-\gamma)^{1/2}}\big).$$

This result shows a dependency on ϵ of $\widetilde{O}(\frac{1}{\epsilon^2})$, compared to $\widetilde{O}(\frac{1}{\epsilon^4})$ from other related ADP algorithms (Munos and Szepesvári, 2008; Antos et al., 2008b). The price that we pay is a multiplicative constant 2 in front of the inherent error $d^\pi_{\mathcal{F}}$. The error from second term on RHS decays exponentially with iterations K. The proof is in Appendix A.5.

We can show an analogous generalization bound for FQI. While FQI has been widely used, to the best of our knowledge, a complete analysis of FQI for non-linear function approximation has not been previously reported.[5]

Definition 2.4.2 (inherent Bellman optimality error)**.** (Munos and Szepesvári, 2008) Recall that the Bellman optimality operator is defined as $(\mathbb{T}Q)(x,a) = r(x,a) +$

[5]FQI for continuous action space from (Antos et al., 2008a) is a variant of fitted policy iteration and not the version of FQI under consideration. The appendix of (Lazaric and Restelli, 2011) contains a proof of FQI specifically for linear function class.

$\gamma \int_{\mathbf{X}} \min_{a' \in \mathbf{A}} Q(x', a') p(dx'|x, a)$. Given a function class $\mathcal{F}$, the *inherent Bellman error* is defined as $d_{\mathcal{F}} = \sup_{g \in \mathcal{F}} \inf_{f \in \mathcal{F}} \|f - \mathbb{T}g\|_{\mu}$, where $\|\cdot\|_{\mu}$ is the ℓ_2 norm weighted by μ, the state-action distribution induced by π_{D}.

Theorem 2.4.3 (Generalization error of FQI)**.** *Under Assumption 1, for $\epsilon > 0$ & $\delta \in (0, 1)$, after K iterations of Fitted Q Iteration, for $n = O\big(\frac{\bar{C}^4}{\epsilon^2}(\log \frac{K}{\delta} + \mathtt{dim}_{\mathcal{F}} \log \frac{\bar{C}^2}{\epsilon^2} + \log \mathtt{dim}_{\mathcal{F}})\big)$, we have with probability $1 - \delta$:*

$$\big|C^* - C(\pi_K)\big| \leqslant \frac{2\gamma}{(1-\gamma)^3}\big(\sqrt{\beta_\mu}\,(2d_{\mathcal{F}} + \epsilon) + 2\gamma^{K/2}\bar{C}\big)$$

where π_K is the policy acting greedy with respect to the returned function Q_K. (Proof in Appendix A.6.)

End-to-End Generalization Guarantee

We are ultimately interested in the test-time performance and constraint satisfaction of the returned policy from Algorithm 2. We now connect the previous analyses from Theorems 2.4.1, 2.4.2 & 2.4.3 into an end-to-end error analysis.

Since Algorithm 2 uses FQI and FQE as subroutines, the inherent Bellman error terms $d_{\mathcal{F}}$ and $d^{\pi}_{\mathcal{F}}$ will enter our overall performance bound. Estimating the inherent Bellman error caused by function approximation is not possible in general (chapter 11 of (Sutton and Barto, 2018b)). We assume existence of a sufficiently expressive $\mathcal{F}$ that can generally make $d_{\mathcal{F}}$ and $d^{\pi}_{\mathcal{F}}$ arbitrarily small. To simplify our end-to-end analysis, consider $d_{\mathcal{F}} = 0$ and $d^{\pi}_{\mathcal{F}} = 0$, i.e., the function class $\mathcal{F}$ is closed under applying the Bellman operator.

Assumption 2 (Bellman operator realizability)**.** *We consider function classes $\mathcal{F}$ sufficiently rich so that $\forall f : \mathbb{T}f \in \mathcal{F}$ & $\mathbb{T}^{\pi} f \in \mathcal{F}$ for the policies π returned by Algorithm 2.*

With Assumptions 1 & 2, we have the following error bound:

Theorem 2.4.4 (Generalization guarantee of Algorithm 2)**.** *Let π^* be the optimal policy to* (OPT)*. Denote $\bar{V} = \bar{C} + B\bar{G}$. Let K be the number of iterations of FQE and FQI. Let $\hat{\pi}$ be the policy returned by Algorithm 2, with termination threshold ω. For $\epsilon > 0$ & $\delta \in (0, 1)$, when $n = O\big(\frac{\bar{V}^4}{\epsilon^2}(\log \frac{K(m+1)}{\delta} + \mathtt{dim}_{\mathcal{F}} \log \frac{\bar{V}^2}{\epsilon^2} + \log \mathtt{dim}_{\mathcal{F}})\big)$, we have with probability at least $1 - \delta$:*

$$C(\hat{\pi}) \leqslant C(\pi^*) + \omega + \frac{(4+B)\gamma}{(1-\gamma)^3}\big(\sqrt{\beta_\mu}\epsilon + 2\gamma^{K/2}\bar{V}\big),$$

and

$$G(\hat{\pi}) \leqslant \tau + 2\frac{\bar{V}+\omega}{B} + \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\big(\sqrt{\beta_\mu}\epsilon + \frac{2\gamma^{K/2}\bar{V}}{(1-\gamma)^{1/2}}\big).$$

The proof is in Appendix A.3. This result guarantees that, upon termination of Algorithm 2, the true performance on the main objective can be arbitrarily close to that of the optimal policy. At the same time, each constraint will be approximately satisfied with high probability, assuming sufficiently large B & K, and sufficiently small ϵ.

2.5 Empirical Study

We perform experiments on two different domains: a grid-world domain (from OpenAI's FrozenLake) under a safety constraint, and a challenging high-dimensional car racing domain (from OpenAI's CarRacing) under multiple behavior constraints. We seek to answer the following questions in our experiments: (i) whether the empirical convergence behavior of Algorithm 2 is consistent with our theory; and (ii) how the returned policy performs with respect to the main objective and constraint satisfaction. Appendix A.8 includes a more detailed discussion of our experiments.

Frozen Lake.

Environment & Data Collection. The environment is an 8x8 grid. The agent has 4 actions N,S,E,W at each state. The main goal is to navigate from a starting position to the goal. Each episode terminates when the agent reaches the goal or falls into a hole. The main cost function is defined as $c = -1$ if goal is reached, otherwise $c = 0$ everywhere. We simulate a non-optimal data gathering policy π_{D} by adding random sub-optimal actions to the shortest path policy from any given state to goal. We run π_D for 5000 trajectories to collect the behavior dataset D (with constraint cost measurement specified below).

Counterfactual Safety Constraint. We augment the main objective c with safety constraint cost defined as $g = 1$ if the agent steps into a hole, and $g = 0$ otherwise. We set the constraint threshold $\tau = 0.1$, roughly 75% of the accumulated constraint cost of behavior policy π_{D}. The threshold can be interpreted as a counterfactually acceptable probability that we allow the learned policy to fail.

Results. The empirical primal dual gap $\widehat{\mathrm{L}}_{\max} - \widehat{\mathrm{L}}_{\min}$ in Figure 2.1 (left) quickly decreases toward the optimal gap of zero. The convergence is fast and monotonic, supporting the predicted behavior from our theory. The test-time performance in Figure 2.1 (middle) shows the safety constraint is always satisfied, while the main

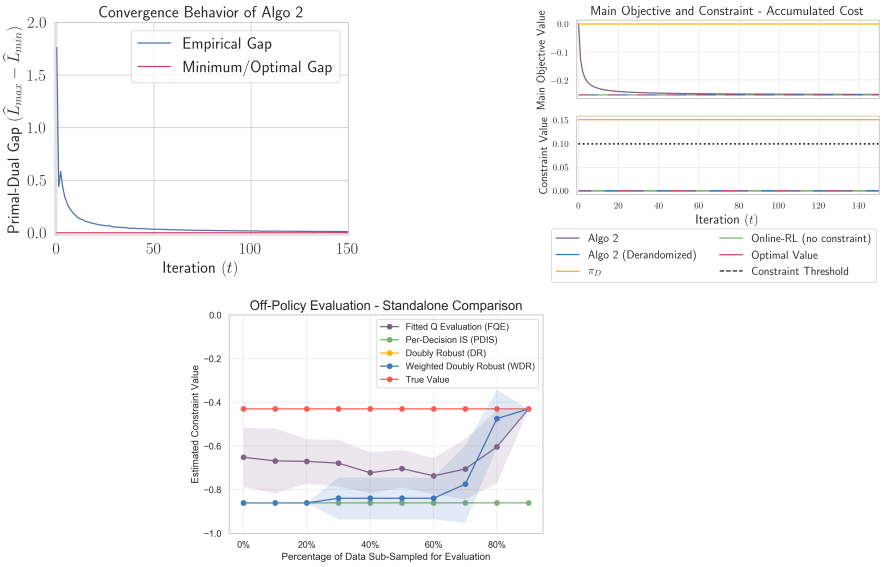


Figure 2.1: *FrozenLake Results*. *(Left)* Empirical duality gap of algorithm 2 vs. optimal gap. *(Middle)* Comparison of returned policy and others w.r.t. (top) main objective value and (bottom) safety constraint value. *(Right)* FQE vs. other OPE methods on a standalone basis.

objective cost also smoothly converges to the optimal value achieved by an online RL baseline (DQN) trained without regard to the constraint. The returned policy significantly outperformed the data gathering policy π_{D} on both main and safety cost.

Car Racing.

Environment & Data Collection. The car racing environment (OpenAI), is a high-dimensional domain where the state is a $96 \times 96 \times 3$ tensor of raw pixels. The action space $\mathbf{A} = \{\text{steering} \times \text{gas} \times \text{brake}\}$ takes 12 discretized values. The goal in this episodic task is to traverse over 95% of the track, measured by a given number of "tiles" as a proxy for distance coverage. The agent receives a reward (negative cost) for each unique tile crossed and no reward if the agent is off-track. A small positive cost applies at every time step, with maximum horizon of 1000 for each episode. With these costs given by the environment, one can train online RL agent using DDQN (Van Hasselt et al., 2016). We collect $\approx$ 5000 trajectories from DDQN's randomization, resulting in data set D with $\approx$ 94000 transition tuples.

Fast Driving under Behavioral Constraints. We study the problem of minimizing environment cost while subject to two behavioral constraints: smooth driving and lane centering. The first constraint G_0 approximates smooth driving by $g_0(x,a) = 1$ if a contains braking action, and 0 otherwise. The second constraint cost g_1 measures

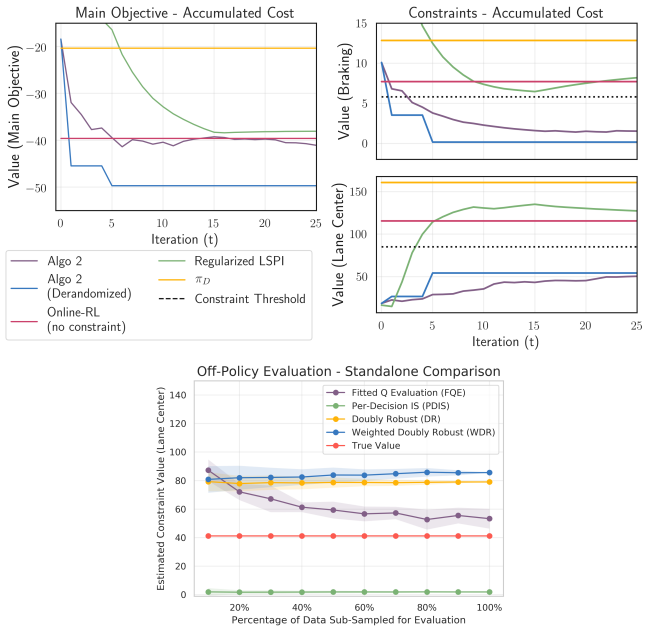


Figure 2.2: *CarRacing Results*. *(Left)* & *(Middle)* (Lower is better) Comparing our algorithm, regularized LSPI, online RL w/o constraints, behavior policy π_D w.r.t. main cost objectives and two constraints. *(Right)* FQE vs. other OPE methods on a standalone basis.

the distance between the agent and center of lane at each time step. This is a highly challenging setup since three objectives and constraints are in direct conflict with one another, e.g., fast driving encourages the agent to cut corners and apply frequent brakes to make turns. Outside of this work, we are not aware of previous work in policy learning with 2 or more constraints in high-dimensional settings.

Baseline and Procedure. As a naïve baseline, DDQN achieves low cost, but exhibits "non-smooth" driving behavior (see our supplementary videos). We set the threshold for each constraint to 75% of the DDQN benchmark. We also compare against regularized batch RL algorithms (Farahmand et al., 2009), specifically regularized LSPI. We instantiate our subroutines, FQE and FQI, with multi-layered CNNs. We augment LSPI's linear policy with non-linear features derived from a well-performing FQI model.

Results. The returned mixture policy from our algorithm achieves low main objective cost, comparable with online RL policy trained without regard to constraints. After several initial iterations violating the braking constraint, the returned policy - corresponding to the appropriate λ trade-off - satisties both constraints, while improving the main objective. The improvement over data gathering policy is

significant for both constraints and main objective.

Regularized policy learning is an alternative approach to (OPT) (section 2.2). We provide the regularized LSPI baseline the same set of λ found by our algorithm for one-shot regularized learning (Figures 2.2 (left & middle)). While regularized LSPI obtains good performance for the main objective, it does not achieve acceptable constraint satisfaction. By default, regularized policy learning requires parameter tuning heuristics. In principle, one can perform a grid-search over a range of parameters to find the right combination - we include such an example for both regularized LSPI and FQI in Appendix A.8. However, since our objective and constraints are in conflict, main objective and constraint satisfaction of policies returned by one-shot regularized learning are sensitive to step changes in λ. In constrast, our approach is systematic, and is able to avoid the curse-of-dimensionality of brute-force search that comes with multiple constraints.

In practice, one may wish to deterministically extract a single policy from the returned mixture for execution. A de-randomized policy can be obtained naturally from our algorithm by selecting the best policy from the existing FQE's estimates of individual `Best-response` policies.

Off-Policy Evaluation

The off-policy evaluation by FQE is critical for updating policies in our algorithm, and is ultimately responsible for certifying constraint satisfaction. While other OPE methods can also be used in place of FQE, we find that the estimates from popular methods are not sufficiently accurate in a high-dimensional setting. As a standalone comparison, we select an individual policy and compare FQE against PDIS (Precup et al., 2000), DR (Jiang and Li, 2016a) and WDR (Thomas and Brunskill, 2016b) with respect to the constraint cost evaluation. To compare both accuracy and data-efficiency, for each domain we randomly sample different subsets of dataset D (from 10% to 100% transitions, 30 trials each). Figure 2.1 (right) and 2.2 (right) illustrate the difference in quality. In the FrozenLake domain, FQE performs competitively with the top baseline method (DR and WDR), converging to the true value estimate as the data subsample grows close to 100%. In the high-dimensional car domain, FQE signficantly outperforms other methods.

2.6 Other Related Work

Constrained MDP (CMDP). Among the most important techniques for solving CMDP are the Lagrangian approach and solving the dual LP program via occupation

measure(Altman, 1999). However, these approaches require known MDP, and small state dimension so that solving via an LP is tractable. More recently, the constrained policy optimization approach (CPO) by (Achiam et al., 2017) learns a policy when the model is not initially known. The focus of CPO is on online safe exploration, and thus is not directly comparable to our setting. Other approaches (Cheng et al., 2019d; Dalal et al., 2018) address safe exploration by building the constraint directly into the policy.

Multi-objective Reinforcement Learning (MORL). (Van Moffaert and Nowé, 2014; Roijers et al., 2013) Approaches to MORL have largely focused on approximating the Pareto frontier that trades-off competing objectives (Van Moffaert and Nowé, 2014; Roijers et al., 2013). The underlying approach to MORL frequently relies on linear or non-linear scalarization of rewards to heuristically turns the problem into a standard RL problem. Our proposed approach represents another systematic paradigm to solve MORL, whether in batch or online settings.

2.7 Discussion

We have presented in this chapter a systematic approach for batch policy learning under multiple constraints. Our problem formulation can accommodate general definition of constraints, as partly illustrated by our experiments. We provide guarantees for our algorithm for both the main objective and constraint satisfaction. Our empirical results show a promise of making constrained batch policy learning applicable for real-world domains, where behavior data is abundant.

While our algorithmic development focuses on the batch setting, i.e.,our implementation complies with the steps laid out in Algorithm 2. Adopting the proposed algorithmic approach to the online setting is relatively straightforward. However, in very large scale or high-dimensional problems, we may consider a noisy update version for both policy learning and evaluation. We leave the theorerical and practical exploration of this extension to future work. In our high-dimensional domain with long horizon, our proposed FQE algorithm for OPE achieves strong results. More extensive comparisons between FQE and other contemporary OPE methods deserve further study. We will focus on the problem of off-policy value estimation in the next chapter.

Chapter 3

OFF-POLICY VALUE ESTIMATION FOR REINFORCEMENT LEARNING

Summary. As seen from the previous chapter, the problem of off-policy value estimation (OPE) is critical for reliable value-based approaches to policy learning. Also known as off-policy policy evaluation in reinforcement learning context, OPE has recently attracted a significant attention due to its practical impoirtance. However, the disparate experimental conditions in recent OPE literature make it difficult both for practitioners to choose a reliable estimator for their application domain, as well as for researchers to identify fruitful research directions. In this chapter, we present the first detailed empirical study of a broad suite of OPE methods. Based on thousands of experiments and empirical analysis, we offer a summarized set of guidelines to advance the understanding of OPE performance in practice, and suggest directions for future research. Along the way, our empirical findings challenge several commonly held beliefs about which class of approaches tends to perform well. Our accompanying software implementation serves as a first comprehensive benchmark for OPE.

3.1 Introduction to Off-Policy Value Estimation

We focus on understanding the relative performance of existing methods for off-policy value estimation (OPE), which is the problem of estimating the value of a target policy using only pre-collected historical data generated by another policy. The earliest OPE methods rely on classical importance sampling to handle the distribution mismatch between the target and behavior policies (Precup et al., 2000). Many advanced OPE methods have since been proposed for both contextual bandits (Dudík et al., 2011b; Bottou et al., 2013; Swaminathan et al., 2017; Wang et al., 2017; Li et al., 2015; Ma et al., 2020) and reinforcement learning settings (Jiang and Li, 2016a; Dudík et al., 2011b; Farajtabar et al., 2018a; Liu et al., 2018c; Xie et al., 2019). These new developments reflect practical interests in deploying reinforcement learning to safety-critical situations (Li et al., 2011; Wiering, 2000; Bottou et al., 2013; Bang and Robins, 2005), and the increasing importance of off-policy learning and counterfactual reasoning more broadly (Degris et al., 2012; Thomas et al., 2017; Munos et al., 2016; Le et al., 2019b; Liu et al., 2019; Nie et al.,

2019). OPE is also closely related to the problem of dynamic treatment regimes in the causal inference literature (Murphy et al., 2001).

Empirical validations have long contributed to the scientific understanding and advancement of machine learning techniques (Chapelle and Li, 2011; Caruana et al., 2008; Caruana and Niculescu-Mizil, 2006). Recently, many have called for careful examination of empirical findings of contemporary deep learning and deep reinforcement learning efforts (Henderson et al., 2018; Locatello et al., 2019). As OPE is central to real-world applications of reinforcement learning, an in-depth empirical understanding is critical to ensure usefulness and accelerate progress. While many recent methods are built on sound mathematical principles, a practitioner is often faced with a non-trivial task of selecting the most appropriate estimator for their application. A notable gap in current literature is a comprehensive empirical understanding of contemporary methods, due in part to the disparate testing environments and varying experimental conditions among prior work. Consequently, there is little holistic insight into where different methods particularly shine, nor a systematic summary of the challenges one may encounter when in different scenarios. Researchers and practitioners may reasonably deduce the following commonly held impressions from surveying the literature:

1. Doubly robust methods are often assumed to outperform direct and importance sampling methods.
2. Horizon length is the primary driver of poor performance for OPE estimators.
3. Model-based is the go-to direct method, either standalone or as part of a doubly-robust estimator.

The reality, as we will discuss, is much more nuanced. In this chapter, we take a closer look at recently proposed methods and offer a thorough empirical study of a wide range of estimators. We design various experimental conditions to explore the success and failure modes of different methods. We synthesize general insights to guide practitioners, and suggest directions for future research. Finally, we provide a highly extensive software package that can interface with new experimental environments and methods to run new OPE experiments at scale.

Notations and Preliminaries for the Chapter

As per RL standard, we represent the environment by $\langle X, A, P, R, \gamma \rangle$. X is the state space (or observation space in the non-Markov case), A is the (finite) action space,

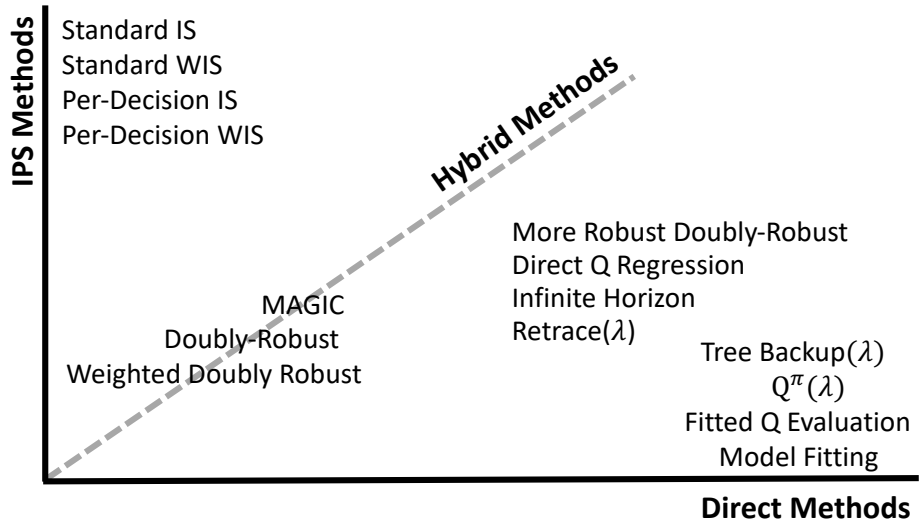


Figure 3.1: *Categorization of OPE methods. Some methods are direct but have IPS influence and thus fit slightly away from the direct methods axis.*

$P : X \times A \times X \to [0,1]$ is the transition function, $R : X \times A \to \mathbb{R}$ is the reward, and discount factor $\gamma \in (0,1]$. A policy π maps states to a distribution over actions, and $\pi(a|x)$ denotes the probability of choosing $a \in A$ in $x \in X$.

OPE is typically considered in the episodic RL setting. A behavior policy π_b generates a historical data set, $D = \{\tau^i\}_{i=1}^N$, of N trajectories (or episodes), where i indexes over trajectories, and $\tau^i = (x_0^i, a_0^i, r_0^i, \ldots, x_{T-1}^i, a_{T-1}^i, r_{T-1}^i)$. The episode length T is frequently assumed to be fixed for notational convenience. In practice, one can pad additional absorbing states to handle variable lengths. Given a desired evaluation policy π_e, the OPE problem is to estimate the value $V(\pi_e)$, defined as:

$$V(\pi_e) = \mathbb{E}_{x\sim d_0}\left[\sum_{t=0}^{T-1} \gamma^t r_t | x_0 = x\right],$$

with $a_t \sim \pi_e(\cdot|x_t)$, $x_{t+1} \sim P(\cdot|x_t, a_t)$, $r_t \sim R(x_t, a_t)$, and d_0 is the initial state distribution.

3.2 Overview of Off-Policy Value Estimators

OPE methods were historically categorized into importance sampling, direct, and doubly robust methods. This demarcation was first introduced for contextual bandits (Dudík et al., 2011b), and later to reinforcement learning (Jiang and Li, 2016a). Some recent methods have blurred the boundary of these categories, such

as Retrace(λ) (Munos et al., 2016) that uses a product of importance weights of multiple time steps for off-policy Q correction, and MAGIC (Thomas and Brunskill, 2016a) that switches between importance weighting and direct methods.

In this section, we propose to regroup OPE into three similar classes of methods, but with expanded definition for each category. Figure 3.1 provides an overview of OPE methods that we consider. The relative positioning of different methods reflects how close they are to being a pure regression-based estimator versus a pure importance sampling-based estimator. Appendix B.4 contains a full description of all methods under consideration.

Inverse Propensity Scoring (IPS)

Inverse Propensity Scoring (IPS), also called importance sampling, is widely used in statistics (Powell and Swann, 1966; Hammersley and Handscomb, 1964; Horvitz and Thompson, 1952) and RL (Precup et al., 2000). The key idea is to reweight the rewards in the historical data by the importance sampling ratio between π_e and π_b, i.e., how likely a reward is under π_e versus π_b. IPS methods yield consistent and (typically) unbiased estimates; however the product of importance weights can be unstable for long time horizons. The cumulative importance weight between π_e and π_b is written as $\rho^i_{j:j'} = \prod_{t=j}^{\min(j',T-1)} \frac{\pi_e(a^i_t|x^i_t)}{\pi_b(a^i_t|x^i_t)}$ (where $\rho^i_{t:t'} = 1$ for $t' < t$). Weighted IPS replaces a normalization factor N by $w_{j:j'} = \frac{1}{N}\sum_{i=1}^{N} \rho^i_{j:j'}$. The weighted versions are biased but strongly consistent.

Importance Sampling (IS) takes the form: $\sum_{i=1}^{N} \frac{\rho^i_{0:T-1}}{N} \sum_{t=0}^{T-1} \gamma^t r_t$. There are three other main IPS variants that we consider: Per-Decision Importance Sampling (PDIS), Weighted Importance Sampling (WIS) and Per-Decision WIS (PDWIS) (see Appendix Table B.14 for full definitions). Other variants of IPS exist but are neither consistent nor unbiased (Thomas, 2015). IPS often assumes known π_b, which may not be possible – one approach is to estimate π_b from data (Hanna et al., 2019), resulting in a potentially biased estimator that can sometimes outperform traditional IPS methods.

Direct Methods (DM)

The main distinction of direct methods from IPS is the focus on regression-based techniques to (more) directly estimate the value functions of the evaluation policy (Q^{π_e} or V^{π_e}). We consider eight different direct approaches, described completely in appendix B.4. Similar to policy learning literature, we can view OPE through the

Table 3.1: Environment parameters - Part 1

Environment	Graph	Graph-MC	MC	Pix-MC	Enduro
Markov?	yes	yes	yes	yes	yes
State/Obs	position	position	[pos, vel]	pixels	pixels
T	4 or 16	250	250	250	1000
Stoch Env?	variable	no	no	no	no
Stoch Rew?	variable	no	no	no	no
Sparse Rew?	variable	terminal	terminal	terminal	dense
$\hat{Q}$ Func. Class	tabular	tabular	linear/NN	NN	NN

Table 3.2: Environment parameters - Part 2

Environment	Graph-POMDP	GW	Pix-GW
Markov?	no	yes	yes
State/Obs	position	position	pixels
T	2 or 8	25	25
Stoch Env?	no	no	variable
Stoch Rew?	no	no	no
Sparse Rew?	terminal	dense	dense
$\hat{Q}$ Func. Class	tabular	tabular	NN

lens of model-based vs. model-free approaches[1].

Model-based. Perhaps the most commonly used DM is model-based (also called approximate model, denoted AM), where the transition dynamics, reward function and termination condition are directly estimated from historical data (Jiang and Li, 2016a; Paduraru, 2013). The resulting learned MDP is then used to compute the value of π_e, e.g., by Monte-Carlo policy evaluation.

Model-free. Estimating the action-value function $\widehat{Q}(\cdot;\theta)$, parameterized by θ, is the focus of several model-free approaches. The value estimate is then: $\widehat{V}(\pi_e) = \frac{1}{N}\sum_{i=1}^{N}\sum_{a\in A}\pi_e(a|s_0^i)\widehat{Q}(x_0^i,a;\theta)$. A simple example is Fitted Q Evaluation (FQE) (Le et al., 2019a), which is a model-free counterpart to AM, and is functionally a policy evaluation counterpart to batch Q learning. FQE learns a sequence of

[1] the distinction is arguably blurry. We stick with this convention simply for linguistic convenience

estimators $\widehat{Q}(\cdot,\theta) = \lim_{k\to\infty} \widehat{Q}_k$, where:

$$\widehat{Q}_k = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} (\widehat{Q}_{k-1}(x_t^i, a_t^i; \theta) - y_t^i)^2,$$
$$y_t^i \equiv r_t^i + \gamma \mathbb{E}_{\pi_e} \widehat{Q}_{k-1}(x_{t+1}^i, \cdot; \theta), \quad \widehat{Q}_0 \equiv 0.$$

Indeed, several model-free methods originated from off-policy learning settings, but are also natural for OPE. $Q^{\pi}(\lambda)$ (Harutyunyan et al., 2016) can be viewed as a generalization of FQE that looks to the horizon limit to incorporate the long-term value into the backup step. Retrace(λ) (Munos et al., 2016) and Tree-Backup(λ) (Precup et al., 2000) also use full trajectories, but additionally incorporate varying levels of clipped importance weights adjustment. The λ-dependent term mitigates instability in the backup step, and is chosen based on experimental findings of Munos et al. (2016).

Q Regression (Q-Reg) and More Robust Doubly-Robust (MRDR) (Farajtabar et al., 2018a) are two recently proposed direct methods that make use of cumulative importance weights in deriving the regression estimate for Q^{π_e}, solved through a quadratic program. MRDR changes the objective of the regression to that of directly minimizing the variance of the Doubly-Robust estimator (see Section 3.2).

Liu et al. (2018c) recently proposed a method for the infinite horizon setting (IH). While IH can be viewed as a Rao-Blackwellization of the IS estimator, we include it in the DM category because it essentially solves the Bellman equation for state distributions and requires function approximation, which are more characteristic of DM. IH shifts the focus from importance sampling over action sequences to estimating the importance ratio ω between *state density distributions* induced by π_b and π_e. This ratio replaces all but the final importance weights ρ_{T-1} in the IH estimate, which resembles IS. More recently, several estimators inspired by density ratio estimation idea have been proposed (Nachum et al., 2019; Uehara and Jiang, 2019; Xie et al., 2019) - we will leave evaluation of these new extensions for future work.

Hybrid Methods (HM)

Hybrid methods subsume doubly robust-like approaches, which combine aspects of both IPS and DM. Standard doubly robust OPE (denoted DR) (Jiang and Li, 2016a) is an unbiased estimator that leverages a DM to decrease the variance of the

unbiased estimates produced by importance sampling techniques:

$$\sum_{i=1}^{N} \frac{\widehat{V}(x_0^i)}{N} + \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \gamma^t \rho_{0:t}^i [r_t^i - \widehat{Q}(x_t^i, a_t^i) + \gamma \widehat{V}(x_{t+1}^i)].$$

Other HMs include Weighted Doubly-Robust (WDR) and MAGIC (see Appendix B.4). WDR self-normalizes the importance weights (similar to WIS). MAGIC introduces adaptive switching between DR and DM; in particular, one can imagine using DR to estimate the value for part of a trajectory and then using DM for the remainder. Using this idea, MAGIC (Thomas and Brunskill, 2016a) finds an optimal linear combination among a set that varies the switch point between WDR and DM. Note that any DM that returns $\widehat{Q}^{\pi_e}(x, a; \theta)$ yields a set of corresponding DR, WDR, and MAGIC estimators. As a result, we consider twenty-one hybrid approaches in our experiments.

3.3 Experiments

Experiment Design Principles. We consider several domain characteristics (simple-complex, deterministic-stochastic, sparse-dense rewards, short-long horizon), π_b, π_e pairs (close-far), and data sizes N (small-large), to study OPE performance under varying conditions.

We use two standard RL benchmarks from OpenAI Brockman et al. (2016): Mountain Car (MC) and Enduro Atari game. As many RL benchmarks are fixed and deterministic, we design 6 additional environments that allow control over various conditions: (i) Graph domain (tabular, varying stochasticity and horizon), (ii) Graph-POMDP (tabular, control for representation), (iii) Graph-MC (simplifying MC to tabular case), (iv) Pixel-MC (study MC in high-dimensional setting), (v) Gridworld (tabular, long horizon version) and (vi) Pixel-Gridworld (controlled Gridworld experiments with function approximation).

All together, our benchmark consists of eight environments with characteristics summarized in Table 3.1. Complete descriptions can be found in Appendix B.5.

Protocol & Metrics. Each experiment depends on specifying environment and its properties, behavior policy π_b, evaluation policy π_e, and number of trajectories N from rolling-out π_b for historical data. The true on-policy value $V(\pi_e)$ is the Monte-Carlo estimate via $10,000$ rollouts of π_e. We repeat each experiment $m = 10$ times with different random seeds. We judge the quality of a method via two metrics:

- Relative mean squared error (*Relative MSE*): $\frac{1}{m}\sum_{i=1}^{m}\frac{(\widehat{V}(\pi_e)_i-\frac{1}{m}\sum_{j=1}^{m}V(\pi_e)_j)^2}{(\frac{1}{m}\sum_{j=1}^{m}V(\pi_e)_j)^2}$, which allows a fair comparison across different conditions.[2]
- *Near-top Frequency*: For each experimental condition, we include the number of times each OPE estimator is within 10% of the best performing estimator to facilitate aggregate comparison across domains.

Implementation & Hyperparameters. With thirty-three different OPE methods considered, we run thousands of experiments across the above eight domains. Hyperparameters are selected based on publication, code release or author consultation. We maintain a consistent set of hyperparameters for each estimator and each environment across experimental conditions (see hyperparameter choice in appendix Table B.23). We create a software package that allows running experiments at scale and easy integration with new domains and techniques for future research. Due to limited space, we will show the results from selected experiment conditions. The complete results, with highlighted best method in each class, are available in the appendix.

3.4 Results

What is the best method?

The first important takeaway is that *there is no clear-cut winner*: no single method or method class is consistently the best performer, as multiple environmental factors can influence the accuracy of each estimator. With that caveat in mind, based on the aggregate top performance metrics, we can recommend the following estimators for each method class (See Table 3.3 and appendix Table B.2).

Inverse propensity scoring (IPS). In practice, weighted importance sampling, which is biased, tends to be more accurate and data-efficient than unbiased basic importance sampling methods. Among the four IPS-based estimators, *PDWIS tends to perform best* (Figure 3.4 left).

Direct methods (DM). Generally, FQE, $Q^{\pi}(\lambda)$*, and IH tend to perform the best among DM* (appendix Table B.2). FQE tends to be more data efficient and is the best method when data is limited (Figure 3.5). $Q^{\pi}(\lambda)$ generalizes FQE to multi-step backup, and works particularly well with more data, but is computationally expensive in complex domains. IH is highly competitive in long horizons and with high policy mismatch in a tabular setting (appendix Tables B.6, B.7). In pixel-based

[2]The performance metric in prior OPE work is typically mean squared error MSE= $\frac{1}{m}\sum_{i=1}^{m}(\widehat{V}(\pi_e)_i - V(\pi_e)_i)^2$

Table 3.3: Method Selection Guidelines - Part 1

Class	Recommendation	When to use
Direct	FQE	Stochastic env, severe policy mismatch
	$Q(\lambda)$	Compute non-issue, moderate policy mismatch
	IH	Long horizon, mild policy mismatch, good kernel
IPS	PDWIS	Short horizon, mild policy mismatch
Hybrid	MAGIC FQE	Severe model misspecification
	MAGIC $Q(\lambda)$	Compute non-issue, severe model misspecification

domains, however, choosing a good kernel function for IH is not straightforward, and IH can underperform other DM (appendix Table B.10). We provide a numerical comparison among direct methods for tabular (appendix Figure B.11) and complex settings (Figure 3.4 center).

Table 3.4: Method Selection Guidelines - Part 2. (For Near-top Frequency, see definition in Section 3.3 and support in Table B.2)

Class	Recommendation	Prototypical env.	Near-top Freq.
Direct	FQE	Graph, MC, Pix-MC	23.7%
	$Q(\lambda)$	GW/Pix-GW	15.0%
	IH	Graph-MC	19.0%
IPS	PDWIS	Graph	4.7%
Hybrid	MAGIC FQE	Graph-POMDP, Enduro	30.0%
	MAGIC $Q(\lambda)$	Graph-POMDP	17.3%

Hybrid methods (HM). With the exception of IH, each DM corresponds to three HM: standard doubly robust (DR), weighted doubly robust (WDR), and MAGIC. *For each DM, its WDR version often outperforms its DR version.* MAGIC can often outperform WDR and DR. However, MAGIC comes with additional hyperparameters, as one needs to specify the set of partial trajectory length to be considered. Unsurprisingly, their performance highly depends on the underlying DM. In our experiments, FQE and $Q^\pi(\lambda)$ are typically the most reliable: *MAGIC with FQE or MAGIC with $Q^\pi(\lambda)$ tend to be among the best hybrid methods* (see appendix Figures B.17 - B.21).

Key drivers of method accuracy

The main reason for the inconsistent performance of estimators is various environmental factors that are inadequately studied from prior work. These coupled factors

often impact accuracy interdependently:

- *Representation mismatch:* Function approximators with insufficient representation power weaken DM, and so do overly rich ones as they cause overfitting (e.g., tabular classes). These issues do not impact IPS. Severe misspecification favors HM and weakens DM.
- *Horizon length:* Long horizons hurt all methods, but especially those dependent on importance weights (including IPS, HM and some DM).
- *Policy mismatch:* Large divergence between π_b and π_e hurts all methods, but tends to favor DM in the small data regime relative to HM and IPS. HM will catch up with DM as data size increases.
- *Bad estimation of unknown behavior policy:*[3] π_b estimation quality depends on the state and action dimensionality, and historical data size. Poor π_b estimates cause HM and IPS to underperform simple DM.
- *Environment / Reward stochasticity:* Stochastic environments hurt the data efficiency of all methods, but favor DM over HM and IPS.

We perform a series of controlled experiments to isolate the impact of these factors. Figure 3.3 shows a typical comparison of the best performing method in each class, under a tabular setting with both short and long horizons, and a large mismatch between π_b and π_e. The particular best method in each class may change depending on the specific conditions. Within each class, a general guideline for method selection is summarized in Table 3.3. The appendix contains the full empirical results of all experiments.

A recipe for method selection

Figure 3.2 summarizes our general guideline for navigating key factors that affect the accuracy of different estimators. To guide the readers through the process, we now dive further into our experimental design to test various factors, and discuss the resulting insights.

Do we potentially have representation mismatch? Representation mismatch comes from two sources: model misspecification and poor generalization. Model misspecification refers to the insufficient representation power of the function class used

[3]Poor estimation of π_b can also be seen as model misspecification. We distinguish the representation issue of π_b from other representation issues related to DM

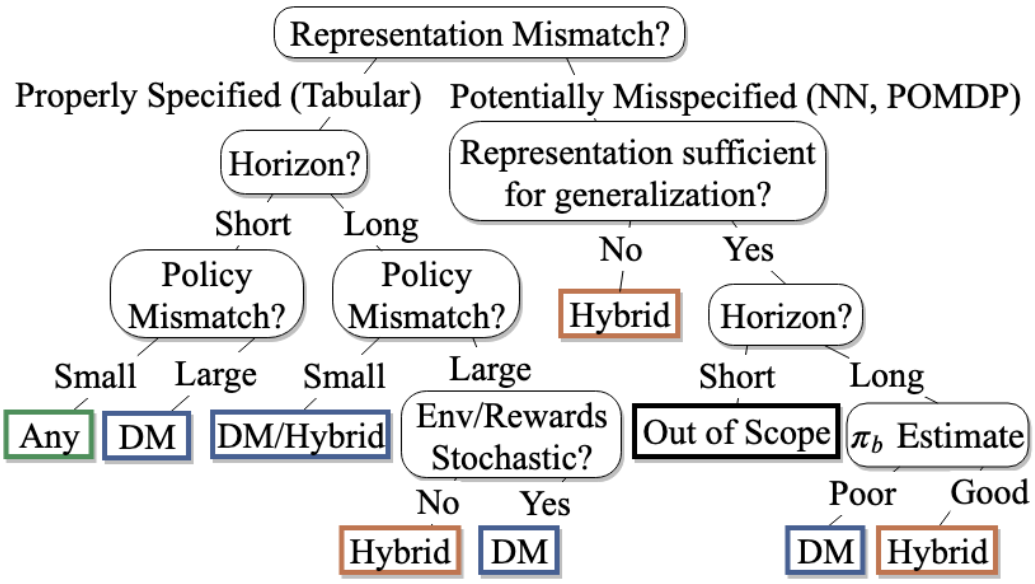


Figure 3.2: *Method Class Selection Decision Tree*. Numerical support can be found in Appendix B.2.

to approximate either the transition dynamics (AM), value function (other DM), or state distribution density ratio (in IH).

Tabular representation for MDP controls for representation mismatch by ensuring adequate function class capacity, as well as zero inherent Bellman error (left branch, Fig 3.2). In such case, we may still suffer from poor generalization without sufficient data coverage, which depends on other factors in the domain settings.

The effect of representation mismatch (right branch, Fig 3.2) can be understood via two controlled scenarios:

- *Misspecified and poor generalization:* We expose the impact of this severe mismatch scenario via the Graph POMDP construction, where selected information are omitted from an otherwise equivalent Graph MDP. HM substantially outperform DM in this setting (Figure 3.3 right versus left).
- *Misspecified but good generalization:* Function class such as neural networks has powerful generalization ability, but may introduce bias and inherent Bellman error[4] (Munos and Szepesvári, 2008; Chen and Jiang, 2019) (see linear vs. neural networks comparison for Mountain Car in appendix Fig B.8). Still, powerful function approximation makes (biased) DM very competitive

[4]defined as $\sup_{g\in \mathrm{F}} \inf_{f\in \mathrm{F}} ||f - \mathbb{T}^{\pi} g||_{d_\pi}$, where F is function class chosen for approximation, and d_π is state distribution induced by evaluation policy π

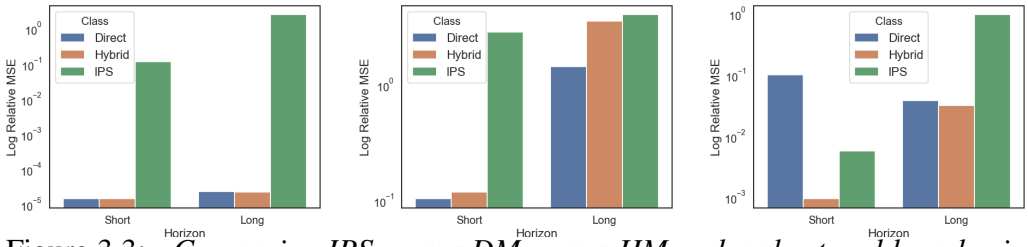


Figure 3.3: *Comparing IPS versus DM versus HM under short and long horizon, large policy mismatch and large data.* Left: *(Graph domain) Deterministic environment.* Center: *(Graph domain) Stochastic environment and rewards.* Right: *(Graph-POMDP) Model misspecification (POMDP). Minimum error per class is shown.*

with HM, especially under limited data and in complex domains (see pixel-Gridworld in appendix Fig B.22-B.24). However, function approximation bias may cause serious problem for high dimensional and long horizon settings. In the extreme case of Enduro (very long horizon and sparse rewards), all DM fail to convincingly outperform a naïve average of behavior data (appendix Fig B.7).

Short horizon vs. Long horizon? It is well-known that IPS methods are sensitive to trajectory length (Li et al., 2015). Long horizon leads to an exponential blow-up of the importance sampling term, and is exacerbated by significant mismatch between π_b and π_e. This issue is inevitable for any unbiased estimator (Jiang and Li, 2016b) (a.k.a., the curse of horizon (Liu et al., 2018b)). Similar to IPS, DM relying on importance weights also suffer from long horizon (appendix Fig B.11), though to a lesser degree. IH aims to bypass the effect of cumulative weighting in long horizons, and indeed performs substantially better than IPS methods in very long horizon domains (Fig 3.4 left).

A frequently ignored aspect in previous OPE work is a proper distinction between fixed, finite horizon tasks (IPS focus), infinite horizon tasks (IH focus), and indefinite horizon tasks, where the trajectory length is finite but varies depending on the policy. Many applications should properly belong to the indefinite horizon category.[5] Applying HM in this setting requires proper padding of the rewards (without altering the value function in the infinite horizon limit) as DR correction typically assumes fixed length trajectories.

[5] Applying IH in the indefinite horizon case requires setting up an absorbing state that loops over itself with zero terminal reward.

How different are behavior and target policies? Similar to IPS, the performance of DM is negatively correlated with the degree of policy mismatch. Figure 3.5 shows the interplay of increasing policy mismatch and historical data size, on the top DM in the deterministic gridworld. We use $(\sup_{a\in A, x\in X} \frac{\pi_e(a|x)}{\pi_b(a|x)})^T$ as an environment-independent metric of mismatch between the two policies. The performance of the top DM (FQE, $Q^\pi(\lambda)$, IH) tend to hold up better than IPS methods when the policy gap increases (appendix Figure B.13). FQE and IH are best in the small data regime, and $Q^\pi(\lambda)$ performs better as data size increases (Figure 3.5). Increased policy mismatch weakens the DM that use importance weights (Q-Reg, MRDR, Retrace(λ) and Tree-Backup(λ)).

Do we have a good estimate of the behavior policy? Often the behavior policy may not be known exactly and requires estimation, which can introduce bias and cause HM to underperform DM, especially in low data regime (e.g., pixel gridworld appendix Figure B.22-B.24). Similar phenomenon was observed in the statistics literature (Kang et al., 2007). As the data size increases, HMs regain the advantage as the quality of the π_b estimate improves.

Is the environment stochastic or deterministic? While stochasticity affects all methods by straining the data requirement, HM are more negatively impacted than DM (Figure 3.3 center, Figure B.12). This can be justified by e.g., the variance analysis of DR, which shows that the variance of the value function with respect to stochastic transitions will be amplified by cumulative importance weights and then contribute to the overall variance of the estimator; see Jiang and Li (2016b, Theorem 1) for further details. We empirically observe that DM frequently outperform their DR versions in the small data case (Figure B.12). In a stochastic environment and tabular setting, HM do not provide significant edge over DM, even in short horizon case. The gap closes as the data size increases (Figure B.12).

Challenging common wisdom

We close this section by briefly revisiting commonly held beliefs about high-level performance of OPE methods.

Are HM always better than DM? No. Overall, DM are surprisingly competitive with HM. Under high-dimensionality, long horizons, estimated behavior policies, or reward/environment stochasticity, HM can underperform simple DM, sometimes significantly (e.g., see appendix Figure B.12).

Concretely, HM can perform worse than DM in the following scenarios that we

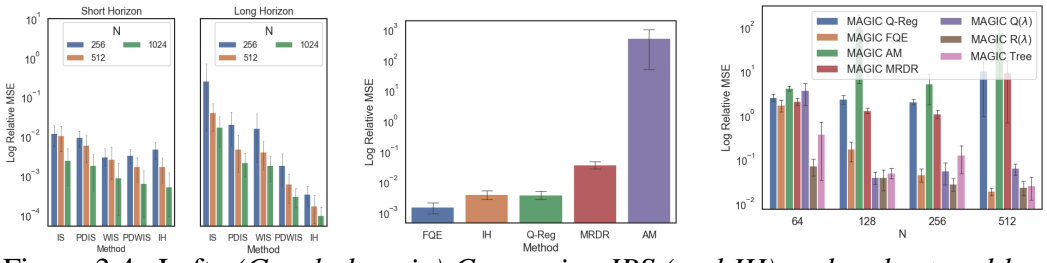


Figure 3.4: Left: *(Graph domain) Comparing IPS (and IH) under short and long horizon. Mild policy mismatch setting. PDWIS is often best among IPS. But IH outperforms in long horizon.* Center: *(Pixel-MC) Comparing direct methods in high-dimensional, long horizon setting. Relatively large policy mismatch. FQE and IH tend to outperform. AM is significantly worse in complex domains. Retrace(λ), Q(λ) and Tree-Backup(λ) are very computationally expensive and thus excluded.* Right: *(Pixel Gridworld) Comparing MAGIC with different base DM and different data size. Large policy mismatch, deterministic environment, known π_b.*

tested:

- Tabular with large policy mismatch, or stochastic environments (appendix Figure B.12, Table B.4, B.7).
- Complex domains with long horizon and unknown behavior policy (appendix Figure B.22-B.24, Table B.9).

When data is sufficient, or model misspecification is severe, HM do provide consistent improvement over DM.

Is horizon length the most important factor? No. Despite conventional wisdom suggesting IPS methods are most sensitive to horizon length, we find that this is not always the case. Policy divergence $\sup_{a\in A,x\in X}\frac{\pi_e(a|x)}{\pi_b(a|x)}$ can be just as, if not more, meaningful. For comparison, we designed two scenarios with identical mismatch $(\sup_{a\in A,x\in X}\frac{\pi_e(a|x)}{\pi_b(a|x)})^T$ as defined in Section 3.4 (see appendix Tables B.12, B.13). Starting from a baseline scenario of short horizon and small policy divergence (appendix Table B.11), extending horizon length leads to $10\times$ degradation in accuracy, while a comparable increase in policy divergence causes a $100\times$ degradation.

How good is model-based direct method (AM)? AM can be among the worst performing direct methods (appendix Table B.2). While AM performs well in tabular setting in the large data case (appendix Figure B.11), it tends to perform poorly in high dimensional settings with function approximation (e.g., Figure 3.4 center). Fitting the transition model $P(x'|x,a)$ is often more prone to small errors than directly approximating $Q(x,a)$. Model fitting errors also compound with long horizons.

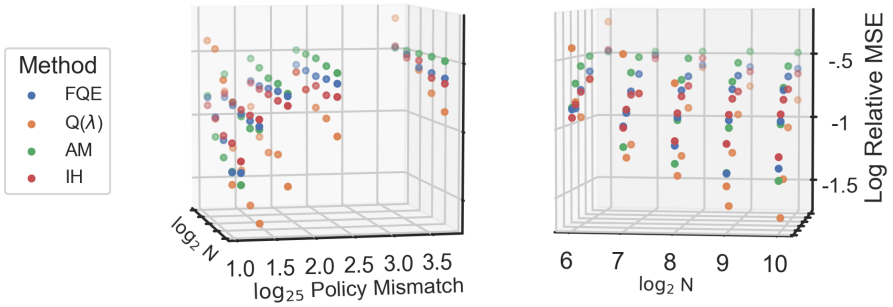


Figure 3.5: *(Gridworld domain) Errors are directly correlated with policy mismatch but inversely correlated with data size. We pick the best direct methods for illustration. The two plots represent the same figure from two different vantage points. See full figures in appendix.*

Other Considerations

Hypeparameter selection. As with many machine learning techniques, hyperparameter choice affects the performance of most estimators (except IPS estimators). The situation is more acute for OPE than the online off-policy learning setting, due to the lack of proper validation signal (such as online game score). When using function approximation, direct methods may not have satisfactory convergence, and require setting a reasonable termination threshold hyperparameter. Q-Reg and MRDR require extra care to avoid ill-conditioning, such as tuning with L1 and L2 regularization.[6] Similarly, the various choice of the kernel function for IH and the index set for hybrid method such as MAGIC have large impact on the performance. *In general, given the choice among different hybrid (or direct) methods, we recommend opting for simplicity as a guiding principle.*

Computational considerations. DM are generally significantly more computationally demanding than IPS. In complex domains, model-free iterative methods can be expensive in training time. Iterative DM that incorporate rollouts until the end of trajectories during training (Retrace(λ), $Q^\pi(\lambda)$, Tree-Backup(λ)) are the most computationally demanding[7], requiring an order of T times the number of $\hat{Q}_{k-1}(x,a)$ lookups per gradient step compared to FQE. Model-based method (AM) are expensive at test time when coupled with HM, since rolling-out the learned model is required at every state along the trajectory.[8] HM versions of direct methods require T times more inference steps, which is often fast after training. In difficult

[6] From correspondence with the authors.

[7] (Munos et al., 2016) limits the rolling-out horizon to 16 in Atari domains, but for the policy learning scenario.

[8] Unlike iterative DM (e.g., FQE), model-based method AM does not benefit from stochastic gradient speedup. Parallelizing the rollouts of AM is highly recommended.

tasks such as Atari games, running AM, Retrace(λ), $Q^\pi(\lambda)$, Tree-Backup(λ) can be prohibitively expensive. Q-Reg, MRDR are non-iterative methods and thus are the fastest to execute among DM. The run-time of IH is dependent on the batch size in building a kernel matrix to compute state similarity. The batch size for IH should be as large as possible, but could significantly slow the training.

Sparsity (non-smoothness) of the rewards: Methods that are dependent on cumulative importance weights are also sensitive to reward sparsity (Figure B.14). We recommend normalizing the rewards. As a rough guideline, zero-centering rewards often improve performance of methods that depend on importance weights. This seemingly naïve practice can be actually viewed as a special case of DR using a constant DM component (baseline), and can yield improvements over vanilla IPS (Jiang and Li, 2016a).

3.5 Discussion and Future Directions

The most difficult environments break all estimators. Atari games pose significant challenges for contemporary techniques due to long horizon and high state dimensionality. It is possible that substantially more historical data is required for current OPE methods to succeed. However, to overcome computational challenge in complex RL domains, it is important to identify principled ways to stabilize iterative methods such as FQE, Retrace(λ), Q(λ) when using function approximation, as convergence is typically not attainable. Some recent progress has been made in stabilizing batch Q-learning in the off-policy learning setting (Fujimoto et al., 2019). It remains to be seen whether similar approach can also benefit DM for OPE.

Lack of short-horizon benchmark in high-dimensional settings. Evaluation of other complex RL tasks with short horizon is currently beyond the scope of our study, due to the lack of a natural benchmark. We refer to prior work on OPE for contextual bandits, which are RL problems with horizon 1 (Dudík et al., 2011b). For contextual bandits, it has been shown that while DR is highly competitive, it is sometimes substantially outperformed by DM (Wang et al., 2017). New benchmark tasks should have longer horizon than contextual bandits, but shorter than typical Atari games. We also currently lack natural stochastic environments in high-dimensional RL benchmarks. An example candidate for medium horizon, complex OPE domain is NLP tasks such as dialogue.

Other OPE settings. Below we outline several practically relevant settings that current literature has overlooked:

- *Continuous actions.* Recent literature on OPE has exclusively focused on finite actions. OPE for continuous action domains will benefit continuous control applications. Currently, continuous action domains will not work with all IPS and HM (see IPS for continuous contextual bandits by Kallus and Zhou (2018)). Among DM, perhaps only FQE may reasonable work with continuous action tasks with some adaptation.
- *Missing data coverage.* A common assumption in the analysis of OPE is a full support assumption: $\pi_e(a|x) > 0$ implies $\pi_b(a|x) > 0$, which often ensure unbiasedness of estimators (Precup et al., 2000; Liu et al., 2018c; Dudík et al., 2011b). This assumption may not hold, and is often not verifiable in practice. Practically, violation of this assumption requires regularization of unbiased estimators to avoid ill-conditioning (Liu et al., 2018c; Farajtabar et al., 2018a). One avenue to investigate is to optimize bias-variance trade-off when the full support is not applicable.
- *Confounding variables.* Existing OPE research often assumes that the behavior policy chooses actions solely based on the state. This assumption is often violated when the decisions in the historical data are made by humans instead of algorithms, who may base their decisions on variables not recorded in the data, causing confounding effects. Tackling this challenge, possibly using techniques from causal inference (Tennenholtz et al., 2019; Oberst and Sontag, 2019), is an important future direction.

Evaluating new OPE estimators. More recently, several new OPE estimators have been proposed: (Nachum et al., 2019; Zhang et al., 2020) further build on the perspective of density ratio estimation from IH; Uehara and Jiang (2019) provides a closely related approach that learns value functions from important ratios; (Xie et al., 2019) proposes improvement over standard IPS by estimating marginalized state distribution in an analogous fashion to IH; (Kallus and Uehara, 2019a,b) analyze double reinforcement learning estimator that makes use of both estimates for Q function and state density ratio. While we have not included these new additions in our analysis, our software implementation is highly modular and can easily accommodate new estimators and environments.

Algorithmic approach to method selection. While we have identified a general guideline for selecting OPE method, often it is not easy to judge whether some decision criteria are satisfied (e.g., quantifying model misspecification, degree of stochasticity, or appropriate data size). As more OPE methods continue to be

developed, an important missing piece is a systematic technique for model selection, given a high degree of variability among existing techniques.

Chapter 4

REGULARIZED LEARNING WITH POLICY-BASED STRUCTURE (SMOOTH IMITATION LEARNING)

Summary. We study the problem of smooth imitation learning for online sequence prediction, where the goal is to train a policy that can smoothly imitate demonstrated behavior in a dynamic and continuous environment in response to online, sequential context input. Since the mapping from context to behavior is often complex, we take a learning reduction approach to reduce smooth imitation learning to a regression problem using complex function classes that are regularized to ensure smoothness. We present a learning meta-algorithm that achieves fast and stable convergence to a good policy. Our approach enjoys several attractive properties, including being fully deterministic, employing an adaptive learning rate that can provably yield larger policy improvements compared to previous approaches, and the ability to ensure stable convergence. Our empirical results demonstrate significant performance gains over previous approaches.

4.1 Introduction

In many complex planning and control tasks, it can be very challenging to explicitly specify a good policy. For such tasks, the use of machine learning to automatically learn a good policy from observed expert behavior, also known as imitation learning or learning from demonstrations, has proven tremendously useful (Abbeel and Ng, 2004; Ratliff et al., 2009; Argall et al., 2009; Ross and Bagnell, 2010; Ross et al., 2011a; Jain et al., 2013).

In this chapter, we study the problem of imitation learning for smooth online sequence prediction in a continuous regime. Online sequence prediction is the problem of making online decisions in response to exogenous input from the environment, and is a special case of reinforcement learning (see Section 4.2). We are further interested in policies that make smooth predictions in a continuous action space.

Our motivating example is the problem of learning smooth policies for automated camera planning (Chen et al., 2016): determining where a camera should look given environment information (e.g., noisy person detections) and corresponding

demonstrations from a human expert.[1]

It is widely accepted that a smoothly moving camera is essential for generating aesthetic video (Gaddam et al., 2015). From a problem formulation standpoint, one key difference between smooth imitation learning and conventional imitation learning is the use of a "smooth" policy class (which we formalize in Section 4.2), and the goal now is to mimic expert demonstrations by choosing the best smooth policy.

The conventional supervised learning approach to imitation learning is to train a classifier or regressor to predict the expert's behavior given training data comprising input/output pairs of contexts and actions taken by the expert. However, the learned policy's prediction affects (the distribution of) future states during the policy's actual execution, and so violates the crucial i.i.d. assumption made by most statistical learning approaches. To address this issue, numerous learning reduction approaches have been proposed (Daumé III et al., 2009; Ross and Bagnell, 2010; Ross et al., 2011a), which iteratively modify the training distribution in various ways such that any supervised learning guarantees provably lift to the sequential imitation setting (potentially at the cost of statistical or computational efficiency).

We present a learning reduction approach to smooth imitation learning for online sequence prediction, which we call SIMILE (**S**mooth **IMI**tation **LE**arning). Building upon learning reductions that employ policy aggregation (Daumé III et al., 2009), we provably lift supervised learning guarantees to the smooth imitation setting and show much faster convergence behavior compared to previous work. Our contributions in this section can be summarized as:

- We formalize the problem of smooth imitation learning for online sequence prediction, and introduce a family of smooth policy classes that is amenable to supervised learning reductions.
- We present a principled learning reduction approach, which we call SIMILE. Our approach enjoys several attractive practical properties, including learning a fully deterministic stationary policy (as opposed to SEARN (Daumé III et al., 2009)), and not requiring data aggregation (as opposed to DAgger (Ross et al., 2011a)) which can lead to super-linear training time.

[1] Access data at `http://www.disneyresearch.com/publication/smooth-imitation-learning/` and code at `http://github.com/hoangminhle/SIMILE`.

- We provide performance guarantees that lift the the underlying supervised learning guarantees to the smooth imitation setting. Our guarantees hold in the agnostic setting, i.e., when the supervised learner might not achieve perfect prediction.
- We show how to exploit a stability property of our smooth policy class to enable adaptive learning rates that yield provably much faster convergence compared to SEARN (Daumé III et al., 2009).
- We empirically evaluate using the setting of smooth camera planning (Chen et al., 2016), and demonstrate the performance gains of our approach.

4.2 Formulating the Problem as Functional Regularization

Let $\mathbf{X} = \{x_1, \ldots, x_T\} \subset \mathcal{X}^T$ denote a context sequence from the environment $\mathcal{X}$, and $\mathbf{A} = \{a_1, \ldots, a_T\} \subset \mathcal{A}^T$ denote an action sequence from some action space $\mathcal{A}$. Context sequence is exogenous, meaning a_t does not influence future context x_{t+k} for $k \geqslant 1$. Let Π denote a policy class, where each $\pi \in \Pi$ generates an action sequence $\mathbf{A}$ in response to a context sequence $\mathbf{X}$. Assume $\mathcal{X} \subset \mathbb{R}^m, \mathcal{A} \subset \mathbb{R}^k$ are continuous and infinite, with $\mathcal{A}$ non-negative and bounded such that $\vec{\mathbf{0}} \preceq a \preceq R\vec{\mathbf{1}} \;\; \forall a \in \mathcal{A}$.

Predicting actions a_t may depend on recent contexts $x_t, \ldots, x_{t-p}$ and actions $a_{t-1}, \ldots, a_{t-q}$. Without loss of generality, we define a state space $\mathcal{S}$ as $\{s_t = [x_t, a_{t-1}]\}$.[2] Policies π can thus be viewed as mapping states $\mathcal{S} = \mathcal{X} \times \mathcal{A}$ to actions $\mathcal{A}$. A roll-out of π given context sequence $\mathbf{X} = \{x_1, \ldots, x_T\}$ is the action sequence $\mathbf{A} = \{a_1, \ldots, a_T\}$:

$$\begin{aligned} a_t &= \pi(s_t) = \pi([x_t, a_{t-1}]), \\ s_{t+1} &= [x_{t+1}, a_t] \quad \forall t \in [1, \ldots, T]\,. \end{aligned}$$

Note that unlike the general reinforcement learning problem, we consider the setting where the state space splits into external and internal components (by definition, a_t influences subsequent states s_{t+k}, but not x_{t+k}). The use of exogenous contexts $\{x_t\}$ models settings where a policy needs to take online, sequential actions based on external environmental inputs, e.g. smooth self-driving vehicles for obstacle avoidance, helicopter aerobatics in the presence of turbulence, or smart grid management for external energy demand. The technical motivation of this dichotomy is that we will enforce smoothness only on the internal state.

[2] We can always concatenate consecutive contexts and actions.

Consider the example of autonomous camera planning for broadcasting a sport event (Chen et al., 2016). $\mathcal{X}$ can correspond to game information such as the locations of the players, the ball, etc., and $\mathcal{A}$ can correspond to the pan-tilt-zoom configuration of the broadcast camera. Manually specifying a good camera policy can be very challenging due to sheer complexity involved with mapping $\mathcal{X}$ to $\mathcal{A}$. It is much more natural to train $\pi \in \Pi$ to mimic observed expert demonstrations. For instance, Π can be the space of neural networks or tree-based ensembles (or both).

Following the basic setup from (Ross et al., 2011a), for any policy $\pi \in \Pi$, let d_t^π denote the distribution of states at time t if π is executed for the first $t-1$ time steps. Furthermore, let $d_\pi = \frac{1}{T}\sum_{t=1}^{T} d_t^\pi$ be the average distribution of states if we follow π for all T steps. The goal of imitation learning is to find a policy $\hat{\pi} \in \Pi$ which minimizes the imitation loss under its own induced distribution of states:

$$\hat{\pi} = \arg\min_{\pi\in\Pi} \ell_\pi(\pi) = \arg\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_\pi}\left[\ell(\pi(s))\right], \tag{4.1}$$

where the (convex) imitation loss $\ell(\pi(s))$ captures how well π imitates expert demonstrations for state s. One common ℓ is squared loss between the policy's decision and the expert demonstration: $\ell(\pi(s)) = \|\pi(s) - \pi^*(s)\|^2$ for some norm $\|.\|$. Note that computing ℓ typically requires having access to a training set of expert demonstrations π^* on some set of context sequences. We also assume an agnostic setting, where the minimizer of (4.1) does not necessarily achieve 0 loss (i.e. it cannot perfectly imitate the expert).

Smooth Imitation Learning & Smooth Policy Class

In addition to accuracy, a key requirement of many continuous control and planning problems is smoothness (e.g., smooth camera trajectories). Generally, "smoothness" may reflect domain knowledge about stability properties or approximate equilibria of a dynamical system. We thus formalize the problem of *smooth imitation learning* as minimizing (4.1) over a smooth policy class Π.

Most previous work on learning smooth policies focused on simple policy classes such as linear models (Abbeel and Ng, 2004), which can be overly restrictive. We instead define a much more general smooth policy class Π as a regularized space of complex models.

Definition 4.2.1 (Smooth policy class Π)**.** *Given a complex model class $\mathcal{F}$ and a class of smooth regularizers $\mathcal{H}$, we define smooth policy class $\Pi \subset \mathcal{F} \times \mathcal{H}$ as satisfying:*

$$\Pi \triangleq \{\pi = (f,h), f \in \mathcal{F}, h \in \mathcal{H} \mid \pi(s) \textit{ is close to both } f(x,a) \textit{ and } h(a) \; \forall \textit{ induced state } s = [x,a] \in \mathcal{S}\}$$

where closeness is controlled by regularization.

For instance, $\mathcal{F}$ can be the space of neural networks or decision trees and $\mathcal{H}$ be the space of smooth analytic functions. Π can thus be viewed as policies that predict close to some $f \in \mathcal{F}$ but are regularized to be close to some $h \in \mathcal{H}$. For sufficiently expressive $\mathcal{F}$, we often have that $\Pi \subset \mathcal{F}$. Thus optimizing over Π can be viewed as constrained optimization over $\mathcal{F}$ (by $\mathcal{H}$), which can be challenging. Our SIMILE approach integrates alternating optimization (between $\mathcal{F}$ and $\mathcal{H}$) into the learning reduction. We provide two concrete examples of Π below.

Example 4.2.1 (Π_λ)**.** Let $\mathcal{F}$ be any complex supervised model class, and define the simplest possible $\mathcal{H} \triangleq \{h(a) = a\}$. Given $f \in \mathcal{F}$, the prediction of a policy π can be viewed as regularized optimization over the action space to ensure closeness of π to both f and h:

$$\begin{aligned}\pi(x,a) &= \arg\min_{a' \in \mathcal{A}} \left\| f(x,a) - a' \right\|^2 + \lambda \left\| h(a) - a' \right\|^2 \\ &= \frac{f(x,a) + \lambda h(a)}{1+\lambda} = \frac{f(x,a) + \lambda a}{1+\lambda},\end{aligned} \tag{4.2}$$

where regularization parameter λ trades-off closeness to f and to previous action. For large λ, $\pi(x,a)$ is encouraged make predictions that stays close to previous action a.

Example 4.2.2 (Linear auto-regressor smooth regularizers)**.** Let $\mathcal{F}$ be any complex supervised model class, and define $\mathcal{H}$ using linear auto-regressors, $\mathcal{H} \triangleq \{h(a) = \theta^\top a\}$, which model actions as a linear dynamical system (Wold, 1939). We can define π analogously to (4.2).

In general, SIMILE requires that Π satisfies a smooth property stated below. This property, which is exploited in our theoretical analysis (see Section 4.5), is motivated by the observation that given a (near) constant stream of context sequence, a stable behavior policy should exhibit a corresponding action sequence with low curvature. The two examples above satisfy this property for sufficiently large λ.

Definition 4.2.2 (H-state-smooth imitation policy)**.** For small constant $0 < H \ll 1$, a policy $\pi([x,a])$ is H-state-smooth if it is H-smooth w.r.t. a, i.e. for fixed $x \in \mathcal{X}$, $\forall a, a' \in \mathcal{A}$, $\forall i$: $\left\|\nabla\pi^i([x,a]) - \nabla\pi^i([x,a'])\right\|_* \leqslant H\|a-a'\|$ where π^i indicates the i^{th} component of vector-valued function[3] $\pi(s) = \left[\pi^1(s),\ldots,\pi^k(s)\right] \in \mathbb{R}^k$, and $\|.\|$ and $\|.\|_*$ are some norm and dual norm respectively. For twice differentiable policy π, this is equivalent to having the bound on the Hessian $\nabla^2\pi^i([x,a]) \preceq H\mathbb{I}_k \ \ \forall i$.

4.3 Related Work in Imitation Learning

The most popular traditional approaches for learning from expert demonstration focused on using approximate policy iteration techniques in the MDP setting (Kakade and Langford, 2002; Bagnell et al., 2003). Most prior approaches operate in discrete and finite action space (He et al., 2012; Ratliff et al., 2009; Abbeel and Ng, 2004; Argall et al., 2009). Some focus on continuous state space (Abbeel and Ng, 2005), but requires a linear model for the system dynamics. In contrast, we focus on learning complex smooth functions within continuous action and state spaces.

One natural approach to tackle the more general setting is to reduce imitation learning to a standard supervised learning problem (Syed and Schapire, 2010; Langford and Zadrozny, 2005; Lagoudakis and Parr, 2003a). However, standard supervised methods assume i.i.d. training and test examples, thus ignoring the distribution mismatch between training and rolled-out trajectories directly applied to sequential learning problems (Kakade and Langford, 2002). Thus a naive supervised learning approach normally leads to unsatisfactory results (Ross and Bagnell, 2010).

Iterative Learning Reductions. State-of-the-art learning reductions for imitation learning typically take an iterative approach, where each training round uses standard supervised learning to learn a policy (Daumé III et al., 2009; Ross et al., 2011a). In each round n, the following happens:

- Given initial state s_0 drawn from the starting distribution of states, the learner executes current policy π_n, resulting in a sequence of states $s_1^n,\ldots,s_T^n$.
- For each s_t^n, a label $\widehat{a}_t^n$ (e.g., expert feedback) is collected indicating what the expert would do given s_t^n, resulting in a new dataset $\mathcal{D}_n = \{(s_t, \widehat{a}_t^n)\}$.

[3]This emphasizes the possibility that π is a vector-valued function of a. The gradient and Hessian are viewed as arrays of k gradient vectors and Hessian matrices of 1-d case, since we simply treat action in $\mathbb{R}^k$ as an array of k standard functions.

- The learner integrates $\mathcal{D}_n$ to learn a policy $\hat{\pi}_n$. The learner updates the current policy to π_{n+1} based on $\hat{\pi}_n$ and π_n.

The main challenge is controlling for the cascading errors caused by the changing dynamics of the system, i.e., the distribution of states in each $\mathcal{D}_n \sim d_{\pi_n}$. A policy trained using d_{π_n} induces a different distribution of states than d_{π_n}, and so is no longer being evaluated on the same distribution as during training. A principled reduction should (approximately) preserve the i.i.d. relationship between training and test examples. Furthermore the state distribution d_π should converge to a stationary distribution.

The arguably most notable learning reduction approaches for imitation learning are SEARN (Daumé III et al., 2009) and DAgger (Ross et al., 2011a). At each round, SEARN learns a new policy $\hat{\pi}_n$ and returns a distribution (or mixture) over previously learned policies: $\pi_{n+1} = \beta\hat{\pi}_n + (1-\beta)\pi_n$ for $\beta \in (0,1)$. For appropriately small choices of β, this stochastic mixing limits the "distribution drift" between π_n and π_{n+1} and can provably guarantee that the performance of π_{n+1} does not degrage significantly relative to the expert demonstrations.[4]

DAgger, on the other hand, achieves stability by aggregating a new dataset at each round to learn a new policy from the combined data set $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_n$. This aggregation, however, significantly increases the computational complexity and thus is not practical for large problems that require many iterations of learning (since the training time grows super-linearly w.r.t. the number of iterations).

Both SEARN and DAgger showed that only a polynomial number of training rounds is required for convergence to a good policy, but with a dependence on the length of horizon T. In particular, to non-trivially bound the total variation distance $\|d_{\pi_{new}} - d_{\pi_{old}}\|_1$ of the state distributions between old and new policies, a learning rate $\beta < \frac{1}{T}$ is required to hold (Lemma 1 of Daumé III, Langford, and Marcu (2009) and Theorem 4.1 of Ross, Gordon, and Bagnell (2011a)). As such, systems with very large time horizons might suffer from very slow convergence.

Our Contributions. Within the context of previous work, our SIMILE approach can be viewed as extending SEARN to smooth policy classes with the following improvements:

[4] A similar approach was adopted in Conservative Policy Iteration for the MDP setting (Kakade and Langford, 2002).

- We provide a policy improvement bound that does not depend on the time horizon T, and can thus converge much faster. In addition, SIMILE has adaptive learning rate, which can further improve convergence.
- For the smooth policy class described in Section 4.2, we show how to generate simulated or "virtual" expert feedback in order to guarantee stable learning. This alleviates the need to have continuous access to a dynamic oracle / expert that shows the learner what to do when it is off-track. In this regard, the way SIMILE integrates expert feedback subsumes the set-up from SEARN and DAgger.
- Unlike SEARN, SIMILE returns fully deterministic policies. Under the continuous setting, deterministic policies are strictly better than stochastic policies as (i) smoothness is critical and (ii) policy sampling requires holding more data during training, which may not be practical for infinite state and action spaces.
- Our theoretical analysis reveals a new sequential prediction setting that yields provably fast convergence, in particular for smooth policy classes on finite-horizon problems. Existing settings that enjoy such results are limited to Markovian dynamics with discounted future rewards or linear model classes.

4.4 Algorithm for Smooth Imitation Learning Setting

Our learning algorithm, called SIMILE (**S**mooth **IMI**tation **LE**arning), is described in Algorithm 4. At a high level, the process can be described as:

1. Start with some initial policy $\hat{\pi}_0$ (Line 2).
2. At iteration n, use π_{n-1} to build a new state distribution $\mathbf{S}_n$ and dataset $\mathcal{D}_n = \{(s_t^n, \widehat{a}_t^n)\}$ (Lines 4-6).
3. Train $\hat{\pi}_n = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim \mathbf{S}_n} [\ell_n(\pi(s))]$, where ℓ_n is the imitation loss (Lines 7-8). Note that ℓ_n needs not be the original ℓ, but simply needs to converge to it.
4. Interpolate $\hat{\pi}_n$ and π_{n-1} to generate a new deterministic policy π_n (Lines 9-10). Repeat from Step 2 with $n \leftarrow n+1$ until some termination condition is met.

Supervised Learning Reduction. The actual reduction is in Lines 7-8, where we follow a two-step procedure of first updating the smooth regularize h_n, and then

Algorithm 4 SIMILE (**S**mooth **IMI**tation **LE**arning)

Input: features $\mathbf{X} = \{x_t\}$, human trajectory $\mathbf{A}^* = \{a_t^*\}$, base routine `Train`, smooth regularizers $h \in \mathcal{H}$

1: Initialize $\mathbf{A}_0 \leftarrow \mathbf{A}^*, \mathbf{S}_0 \leftarrow \{\left[x_t, a_{t-1}^*\right]\}$,
$$h_0 = \arg\min_{h \in \mathcal{H}} \sum_{t=1}^{T} \left\| a_t^* - h(a_{t-1}^*) \right\|$$
2: Initial policy $\pi_0 = \hat{\pi}_0 \leftarrow \texttt{Train}(\mathbf{S}_0, \mathbf{A}_0 | \ h_0)$
3: **for** $n = 1, \ldots, N$ **do**
4: $\quad \mathbf{A}_n = \{a_t^n\} \leftarrow \pi_{n-1}(\mathbf{S}_{n-1})$ *//sequential roll-out*
5: $\quad \mathbf{S}_n \leftarrow \{s_t^n = \left[x_t, a_{t-1}^n\right]\}$ *//* $s_t^n = \left[x_{t:t-p}, a_{t-1:t-q}\right]$
6: $\quad \widehat{\mathbf{A}}_n = \{\widehat{a}_t^n\} \ \forall s_t^n \in \mathbf{S}_n$ *// collect smooth feedback*
7: $\quad h_n = \arg\min_{h \in \mathcal{H}} \sum_{t=1}^{T} \left\| \widehat{a}_t^n - h(\widehat{a}_{t-1}^n) \right\|$ *//new regularizer*
8: $\quad \hat{\pi}_n \leftarrow \texttt{Train}(\mathbf{S}_n, \widehat{\mathbf{A}}_n | \ h_n)$ *// train policy*
9: $\quad \beta \leftarrow \beta(\ell(\hat{\pi}_n), \ell(\pi_{n-1}))$ *//adaptively set* β
10: $\quad \pi_n = \beta\hat{\pi}_n + (1-\beta)\pi_{n-1}$ *// update policy*
11: **end for**

output Last policy π_N

training $\hat{\pi}_n$ via supervised learning. In other words, `Train` finds the best $f \in \mathcal{F}$ possible for a fixed h_n. We discuss how to set the training targets $\widehat{a}_t^n$ below.

Policy Update. The new policy π_n is a deterministic interpolation between the previous π_{n-1} and the newly learned $\hat{\pi}_n$ (Line 10). In contrast, for SEARN, π_n is a stochastic interploation (Daumé III et al., 2009). Lemma 4.5.2 and Corollary 4.5.3 show that deterministic interpolation converges at least as fast as stochastic for smooth policy classes.

This interpolation step plays two key roles. First, it is a form of myopic or greedy online learning. Intuitively, rolling out π_n leads to incidental exploration on the mistakes of π_n, and so each round of training is focused on refining π_n. Second, the interpolation in Line 10 ensures a slow drift in the distribution of states from round to round, which preserves an approximate i.i.d. property for the supervised regression subroutine and guarantees convergence.

However this model interpolation creates an inherent tension between maintaining approximate i.i.d. for valid supervised learning and more aggressive exploration (and thus faster convergence). For example, SEARN's guarantees only apply for small $\beta < 1/T$. SIMILE circumvents much of this tension via a policy improvement bound that allows β to adaptively increase depending on the quality of $\hat{\pi}_n$ (see Theorem 4.5.6), which thus guarantees a valid learning reduction while substantially

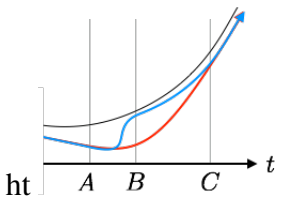


Figure 4.1: Illustration of a smooth simulated feedback (red) vs non-smooth feedback (blue)

speeding up convergence.

Feedback Generation. We can generate training targets $\widehat{a}_t^n$ using "virtual" feedback from simulating expert demonstrations, which has two benefits. First, we need not query the expert π^* at every iteration (as done in DAgger (Ross et al., 2011a)). Continuously acquiring expert demonstrations at every round can be seen as a special case and a more expensive strategy. Second, virtual feedback ensures stable learning, i.e., every $\hat{\pi}_n$ is a feasible smooth policy.

Consider Figure 4.1, where our policy π_n (blue/red) made a mistake at location A, and where we have only a single expert demonstration from π^* (black). Depending on the smoothness requirements of the policy class, we can simulate virtual expert feedback as via either the red line (more smooth) or blue (less smooth) as a tradeoff between squared imitation loss and smoothness.

When the roll-out of π_{n-1} (i.e. $\mathbf{A}_n$) differs substantially from $\mathbf{A}^*$, especially during early iterations, using smoother feedback (red instead of blue) can result in more stable learning. We formalize this notion for Π_λ in Proposition 4.5.8. Intuitively, whenever π_{n-1} makes a mistake, resulting in a "bad" state s_t^n, the feedback should recommend a smooth correction $\widehat{a}_t^n$ w.r.t. $\mathbf{A}_n$ to make training "easier" for the learner.[5] The virtual feedback $\widehat{a}_t^n$ should converge to the expert's action a_t^*. In practice, we use $\widehat{a}_t^n = \sigma a_t^n + (1-\sigma)a_t^*$ with $\sigma \to 0$ as n increases (which satisfies Proposition 4.5.8).

4.5 Theoretical Analysis of Reduction-Based Algorithm

All proofs are deferred to the supplementary material.

Stability Conditions

One natural smoothness condition is that $\pi([x, a])$ should be stable w.r.t. a if x is fixed. Consider the camera planning setting: the expert policy π^* should have

[5] A similar idea was proposed (He et al., 2012) for DAgger-type algorithm, albeit only for linear model classes.

very small curvature, since constant inputs should correspond to constant actions. This motivates Definition 4.2.2, which requires that Π has low curvature given fixed context. We also show that smooth policies per Definition 4.2.2 lead to stable actions, in the sense that "nearby" states are mapped to "nearby" actions. The following helper lemma is useful:

Lemma 4.5.1. *For a fixed x, define $\pi([x,a]) \triangleq \varphi(a)$. If φ is non-negative and H-smooth w.r.t. a., then:*

$$\forall a, a' : \ \left(\varphi(a) - \varphi(a')\right)^2 \leqslant 6H \left(\varphi(a) + \varphi(a')\right) \left\|a - a'\right\|^2 .$$

Writing π as $\pi([x,a]) \triangleq \left[\pi^1([x,a]), \ldots, \pi^k([x,a])\right]$ with each $\pi^i([x,a])$ H-smooth, Lemma 4.5.1 implies $\|(\pi([x,a]) - \pi([x,a']))\| \leqslant \sqrt{12HR}\,\|a - a'\|$ for R upper bounding $\mathcal{A}$. Bounded action space means that a sufficiently small H leads to the following stability conditions:

Condition 1 (Stability Condition 1)**.** Π *satisfies the Stability Condition 1 if for a fixed input feature x, the actions of π in states $s = [x,a]$ and $s' = [x,a']$ satisfy $\|\pi(s) - \pi(s')\| \leqslant \|a - a'\|$ for all $a, a' \in \mathcal{A}$.*

Condition 2 (Stability Condition 2)**.** Π *satisfies Stability Condition 2 if each π is γ-Lipschitz continuous in the action component a with $\gamma < 1$. That is, for a fixed x the actions of π in states $s = [x,a]$ and $s' = [x,a']$ satisfy $\|\pi(s) - \pi(s')\| \leqslant \gamma \|a - a'\|$ for all $a, a' \in \mathcal{A}$.*

These two conditions directly follow from Lemma 4.5.1 and assuming sufficiently small H. Condition 2 is mildly stronger than Condition 1, and enables proving much stronger policy improvement compared to previous work.

Deterministic versus Stochastic

Given two policies π and $\hat{\pi}$, and interpolation parameter $\beta \in (0,1)$, consider two ways to combine policies:

1. **stochastic**: $\pi_{sto}(s) = \hat{\pi}(s)$ with probability β, and $\pi_{sto}(s) = \pi(s)$ with probability $1 - \beta$
2. **deterministic**: $\pi_{det}(s) = \beta\hat{\pi}(s) + (1-\beta)\pi(s)$

Previous learning reduction approaches only use stochastic interpolation (Daumé III et al., 2009; Ross et al., 2011a), whereas SIMILE uses deterministic. The following

result shows that deterministic and stochastic interpolation yield the same expected behavior for smooth policy classes.

Lemma 4.5.2. *Given any starting state s_0, sequentially execute π_{det} and π_{sto} to obtain two separate trajectories $\boldsymbol{A} = \{a_t\}_{t=1}^T$ and $\tilde{\boldsymbol{A}} = \{\tilde{a}_t\}_{t=1}^T$ such that $a_t = \pi_{det}(s_t)$ and $\tilde{a}_t = \pi_{sto}(\tilde{s}_t)$, where $s_t = [x_t, a_{t-1}]$ and $\tilde{s}_t = [x_t, \tilde{a}_{t-1}]$. Assuming the policies are stable as per Condition 1, we have $\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] = a_t \ \forall t = 1, \ldots, T$, where the expectation is taken over all random roll-outs of π_{sto}.*

Lemma 4.5.2 shows that deterministic policy combination (SIMILE) yields unbiased trajectory roll-outs of stochastic policy combination (as done in SEARN & CPI). This represents a major advantage of SIMILE, since the number of stochastic roll-outs of π_{sto} to average to the deterministic trajectory of π_{det} is polynomial in the time horizon T, leading to much higher computational complexity. Furthermore, for convex imitation loss $\ell_\pi(\pi)$, Lemma 4.5.2 and Jensen's inequality yield the following corollary, which states that under convex loss, deterministic policy performs at least no worse than stochastic policy in expectation:

Corollary 4.5.3 (Deterministic Policies Perform Better)**.** *For deterministic π_{det} and stochastic π_{sto} interpolations of two policies π and $\hat{\pi}$, and convex loss ℓ, we have:*

$$\begin{aligned} \ell_{\pi_{det}}(\pi_{det}) &= \ell_{\pi_{sto}}(\mathbb{E}[\pi_{sto}]) \\ &\leqslant \mathbb{E}\left[\ell_{\pi_{sto}}(\pi_{sto})\right] \end{aligned}$$

where the expectation is over all roll-outs of π_{sto}.

Remark. We construct a simple example to show that Condition 1 may be necessary for iterative learning reductions to converge. Consider the case where contexts $\mathbf{X} \subset \mathbb{R}$ are either constant or vary neglibly. Expert demonstrations should be constant $\pi^*([x_n, a^*]) = a^*$ for all n. Consider an unstable policy π such that $\pi(s) = \pi([x, a]) = ka$ for fixed $k > 1$. The rolled-out trajectory of π *diverges* π^* at an exponential rate. Assume optimistically that $\hat{\pi}$ learns the correct expert behavior, which is simply $\hat{\pi}(s) = \hat{\pi}([x, a]) = a$. For any $\beta \in (0, 1)$, the updated policy $\pi' = \beta\hat{\pi} + (1 - \beta)\pi$ becomes $\pi'([x, a]) = \beta a + (1 - \beta)ka$. Thus the sequential roll-out of the new policy π' will also yield an exponential gap from the correct policy. By induction, the same will be true in all future iterations.

Policy Improvement

Our policy improvement guarantee builds upon the analysis from SEARN (Daumé III et al., 2009), which we extend to using adaptive learning rates β. We first restate the main policy improvement result from Daumé III et al. (2009).

Lemma 4.5.4 (SEARN's policy nondegradation - Lemma 1 from Daumé III et al. (2009))**.** *Let $\ell_{max} = \sup_{\pi,s} \ell(\pi(s))$, π' is defined as π_{sto} in lemma 4.5.2. Then for $\beta \in (0, 1/T)$:*

$$\ell_{\pi'}(\pi') - \ell_{\pi}(\pi) \leqslant \beta T \mathbb{E}_{s\sim d_\pi}\left[\ell(\hat{\pi}(s))\right] + \frac{1}{2}\beta^2 T^2 \ell_{max}.$$

SEARN guarantees that the new policy π' does not degrade from the expert π^* by much only if $\beta < 1/T$. Analyses of SEARN and other previous iterative reduction methods (Ross et al., 2011a; Kakade and Langford, 2002; Bagnell et al., 2003; Syed and Schapire, 2010) rely on bounding the variation distance between d_π and $d_{\pi'}$. Three drawbacks of this approach are: (i) non-trivial variation distance bound typically requires β to be inversely proportional to time horizon T, causing slow convergence; (ii) not easily applicable to the continuous regime; and (iii) except under MDP framework with discounted infinite horizon, previous variation distance bounds do not guarantee monotonic policy improvements (i.e. $\ell_{\pi'}(\pi') < \ell_\pi(\pi)$).

We provide two levels of guarantees taking advantage of Stability Conditions 1 and 2 to circumvent these drawbacks. Assuming the Condition 1 and convexity of ℓ, our first result yields a guarantee comparable with SEARN.

Theorem 4.5.5 (T-dependent Improvement)**.** *Assume ℓ is convex and L-Lipschitz, and Condition 1 holds. Let $\epsilon = \max_{s\sim d_\pi} \|\hat{\pi}(s) - \pi(s)\|$. Then:*

$$\ell_{\pi'}(\pi') - \ell_{\pi}(\pi) \leqslant \beta\epsilon LT + \beta\left(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)\right). \tag{4.3}$$

In particular, choosing $\beta \in (0, 1/T)$ yields:

$$\ell_{\pi'}(\pi') - \ell_{\pi}(\pi) \leqslant \epsilon L + \beta\left(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)\right). \tag{4.4}$$

Similar to SEARN, Theorem 4.5.5 also requires $\beta \in (0, 1/T)$ to ensure the RHS of (4.4) stays small. However, note that the reduction term $\beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi))$ allows the bound to be strictly negative if the policy $\hat{\pi}$ trained on d_π significantly improves on $\ell_\pi(\pi)$ (i.e., guaranteed policy improvement). We observe empirically that this often happens, especially in early iterations of training.

Under the mildly stronger Condition 2, we remove the dependency on the time horizon T, which represents a much stronger guarantee compared to previous work.

Theorem 4.5.6 (Policy Improvement)**.** *Assume ℓ is convex and L-Lipschitz continuous, and Condition 2 holds. Let $\epsilon = \max_{s \sim d_\pi} \|\hat{\pi}(s) - \pi(s)\|$. Then for $\beta \in (0,1)$:*

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant \frac{\beta\gamma\epsilon L}{(1-\beta)(1-\gamma)} + \beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)). \tag{4.5}$$

Corollary 4.5.7 (Monotonic Improvement)**.** *Following the notation from Theorem 4.5.6, let $\Delta = \ell_\pi(\pi) - \ell_\pi(\hat{\pi})$ and $\delta = \frac{\gamma\epsilon L}{1-\gamma}$. Then choosing step size $\beta = \frac{\Delta-\delta}{2\Delta}$, we have:*

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant -\frac{(\Delta-\delta)^2}{2(\Delta+\delta)}. \tag{4.6}$$

The terms ϵ and $\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)$ on the RHS of (4.4) and (4.5) come from the learning reduction, as they measure the "distance" between $\hat{\pi}$ and π on the state distribution induced by π (which forms the dataset to train $\hat{\pi}$). In practice, both terms can be empirically estimated from the training round, thus allowing an estimate of β to minimize the bound.

Theorem 4.5.6 justifies using an adaptive and more aggressive interpolation parameter β to update policies. In the worst case, setting β close to 0 will ensure the bound from (4.5) to be close to 0, which is consistent with SEARN's result. More generally, monotonic policy improvement can be guaranteed for appropriate choice of β, as seen from Corollary 4.5.7. This strict policy improvement was not possible under previous iterative learning reduction approaches such as SEARN and DAgger, and is enabled in our setting due to exploiting the smoothness conditions.

Smooth Feedback Analysis

Smooth Feedback Does Not Hurt: Recall from Section 4.4 that one way to simulate "virtual" feedback for training a new $\hat{\pi}$ is to set the target $\hat{a}_t = \sigma a_t + (1-\sigma)a_t^*$ for $\sigma \in (0,1)$, where smooth feedback corresponds to $\sigma \to 1$. To see that simulating smooth "virtual" feedback target does not hurt the training progress, we alternatively view SIMILE as performing gradient descent in a smooth function space (Mason et al., 1999). Define the cost functional $C : \Pi \to \mathbb{R}$ over policy space to be the average imitation loss over $\mathcal{S}$ as $C(\pi) = \int_{\mathcal{S}} \|\pi(s) - \pi^*(s)\|^2 \, dP(s)$. The gradient (Gâteaux derivative) of $C(\pi)$ w.r.t. π is:

$$\nabla C(\pi)(s) = \frac{\partial C(\pi + \alpha\delta_s)}{\partial \alpha}\Big|_{\alpha=0} = 2(\pi(s) - \pi^*(s)),$$

where δ_s is Dirac delta function centered at s. By first order approximation $C(\pi') = C(\beta\hat{\pi} + (1-\beta)\pi) = C(\pi + \beta(\hat{\pi} - \pi)) \approx C(\pi) + \beta\langle \nabla C(\pi), \hat{\pi} - \pi \rangle$. Like traditional gradient descent, we want to choose $\hat{\pi}$ such that the update moves the functional along the direction of negative gradient. In other words, we want to learn $\hat{\pi} \in \Pi$ such that $\langle \nabla C(\pi), \hat{\pi} - \pi \rangle \ll 0$. We can evaluate this inner product along the states induced by π. We thus have the estimate:

$$\begin{aligned}\langle \nabla C(\pi), \hat{\pi} - \pi \rangle &\approx \frac{2}{T}\sum_{t=1}^{T}(\pi(s_t) - \pi^*(s_t))(\hat{\pi}(s_t) - \pi(s_t)) \\ &= \frac{2}{T}\sum_{t=1}^{T}(a_t - a_t^*)(\hat{\pi}([x_t, a_{t-1}]) - a_t).\end{aligned}$$

Since we want $\langle \nabla C(\pi), \hat{\pi} - \pi \rangle < 0$, this motivates the construction of new data set $\mathcal{D}$ with states $\{[x_t, a_{t-1}]\}_{t=1}^T$ and labels $\{\widehat{a}_t\}_{t=1}^T$ to train a new policy $\hat{\pi}$, where we want $(a_t - a_t^*)(\widehat{a}_t - a_t) < 0$. A sufficient solution is to set target $\widehat{a}_t = \sigma a_t + (1-\sigma)a_t^*$ (Section 4.4), as this will point the gradient in negative direction, allowing the learner to make progress.

Smooth Feedback is Sometimes Necessary: When the current policy performs poorly, smooth virtual feedback may be required to ensure stable learning, i.e. producing a feasible smooth policy at each training round. We formalize this notion of feasibility by considering the smooth policy class Π_λ in Example 4.2.1. Recall that smooth regularization of Π_λ via $\mathcal{H}$ encourages the next action to be close to the previous action. Thus a natural way to measure smoothness of $\pi \in \Pi_\lambda$ is via the average first order difference of consecutive actions $\frac{1}{T}\sum_{t=1}^{T}\|a_t - a_{t-1}\|$. In particular, we want to explicitly constrain this difference relative to the expert trajectory $\frac{1}{T}\sum_{t=1}^{T}\|a_t - a_{t-1}\| \leqslant \eta$ at each iteration, where $\eta \propto \frac{1}{T}\sum_{t=1}^{T}\left\|a_t^* - a_{t-1}^*\right\|$.

When π performs poorly, i.e. the "average gap" between current trajectory $\{a_t\}$ and $\{a_t^*\}$ is large, the training target for $\hat{\pi}$ should be lowered to ensure learning a smooth policy is feasible, as inferred from the following proposition. In practice, we typically employ smooth virtual feedback in early iterations when policies tend to perform worse.

Proposition 4.5.8. *Let ω be the average supervised training error from $\mathcal{F}$, i.e. $\omega = \min_{f\in\mathcal{F}} \mathbb{E}_{x\sim\mathcal{X}}[\|f([x,0]) - a^*\|]$. Let the rolled-out trajectory of current policy π be $\{a_t\}$. If the average gap between π and π^* is such that $\mathbb{E}_{t\sim Uniform[1:T]}[\|a_t^* - a_{t-1}\|] \geqslant 3\omega + \eta(1+\lambda)$, then using $\{a_t^*\}$ as feedback will cause the trained policy $\hat{\pi}$ to be*

non-smooth, i.e.:

$$\mathbb{E}_{t\sim Uniform[1:T]}\left[\|\hat{a}_t - \hat{a}_{t-1}\|\right] \geqslant \eta, \tag{4.7}$$

for $\{\hat{a}_t\}$ *the rolled-out trajectory of* $\hat{\pi}$.

4.6 Experiments

Automated Camera Planning. We evaluate SIMILE in a case study of automated camera planning for sport broadcasting (Chen and Carr, 2015; Chen et al., 2016). Given noisy tracking of players as raw input data $\{x_t\}_{t=1}^T$, and demonstrated pan camera angles from professional human operator as $\{a_t^*\}_{t=1}^T$, the goal is to learn a policy π that produces trajectory $\{a_t\}_{t=1}^T$ that is both smooth and accurate relative to $\{a_t^*\}_{t=1}^T$. Smoothness is critical in camera control: fluid movements which maintain adequate framing are preferable to jittery motions which constantly pursue perfect tracking (Gaddam et al., 2015). In this setting, time horizon T is the duration of the event multiplied by rate of sampling. Thus T tends to be very large.

Smooth Policy Class. We use a smooth policy class following Example 4.2.2: regression tree ensembles $\mathcal{F}$ regularized by a class of linear autoregressor functions $\mathcal{H}$ (Chen et al., 2016). See Appendix C.2 for more details.

Summary of Results.

- Using our smooth policy class leads to dramatically smoother trajectories than not regularizing using $\mathcal{H}$.
- Using our adaptive learning rate leads to much faster convergence compared to conservative learning rates from SEARN (Daumé III et al., 2009).
- Using smooth feedback ensures stable learning of smooth policies at each iteration.
- Deterministic policy interpolation performs better than stochastic interpolation used in SEARN.

Smooth versus Non-Smooth Policy Classes. Figure 4.2 shows a comparison of using a smooth policy class versus a non-smooth one (e.g., not using $\mathcal{H}$). We see that our approach can reliably learn to predict trajectories that are both smooth and accurate.

Adaptive vs. Fixed β: One can, in principle, train using SEARN, which requires a very conservative β in order to guarantee convergence. In contrast, SIMILE adaptively selects β based on relative empirical loss of π and $\hat{\pi}$ (Line 9 of Algorithm 4). Let $\texttt{error}(\hat{\pi})$ and $\texttt{error}(\pi)$ denote the mean-squared errors of rolled-out

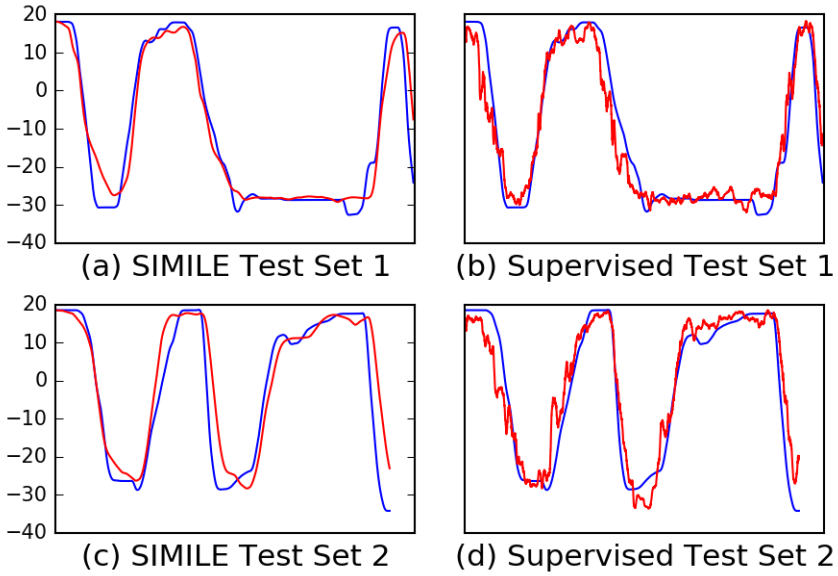


Figure 4.2: Expert (blue) and predicted (red) camera pan angles. Left: SIMILE with <10 iterations. Right: non-smooth policy.

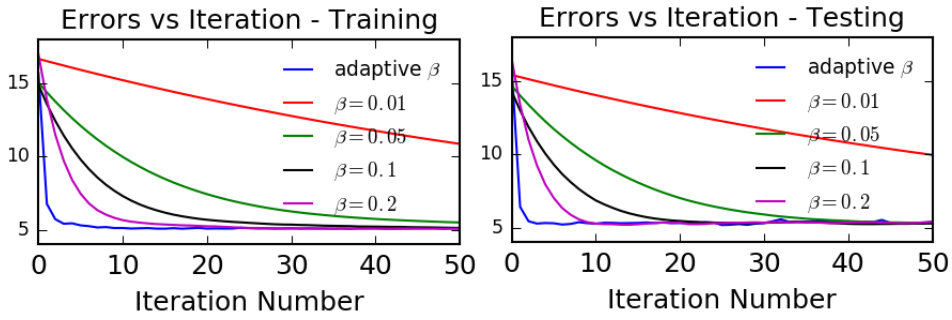


Figure 4.3: Adaptive versus fixed interpolation parameter β.

trajectories $\{\hat{a}_t\}$, $\{a_t\}$, respectively, w.r.t. ground truth $\{a_t^*\}$. We can set β as:

$$\hat{\beta} = \frac{\texttt{error}(\pi)}{\texttt{error}(\hat{\pi}) + \texttt{error}(\pi)}, \tag{4.8}$$

which encourages the learner to disregard bad policies when interpolating, thus allowing fast convergence to a good policy (see Theorem 4.5.6). Figure 4.3 compares the convergence rate of SIMILE using adaptive β versus conservative fixed values of β commonly used in SEARN (Daumé III et al., 2009). We see that adaptively choosing β enjoys substantially faster convergence. Note that very large fixed β may overshoot and worsen the combined policy after a few initial improvements.

Smooth Feedback Generation: We set the target labels to $\hat{a}_t^n = \sigma a_t^n + (1-\sigma)a_t^*$ for $0 < \sigma < 1$ (Line 6 of Algorithm 4). Larger σ corresponds to smoother ($\hat{a}_t^n$

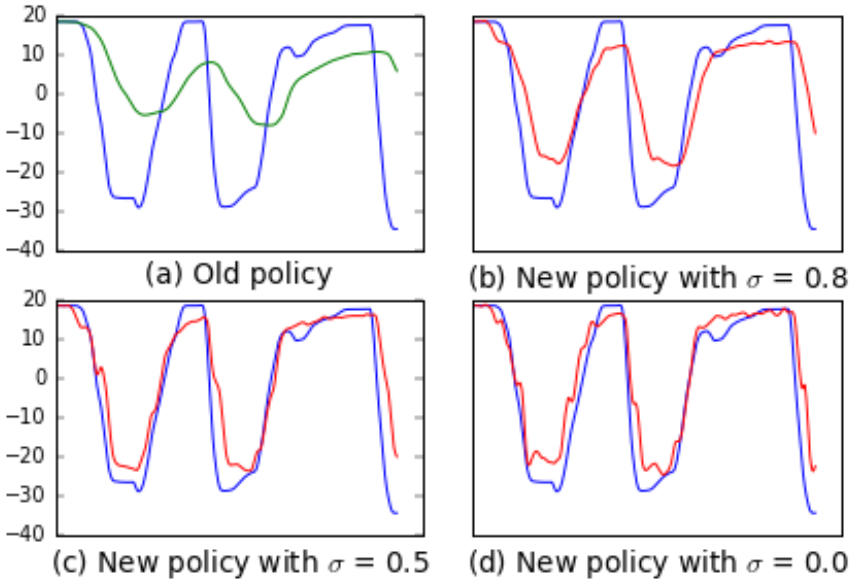
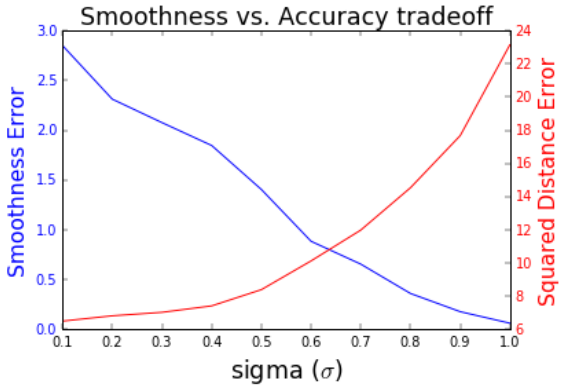


Figure 4.4: Comparing different values of σ.



ht

Figure 4.5: Smoothness Structure vs Accuracy Trade-off

is closer to a^n_{t-1}) but less accurate target (further from a^*_t), as seen in Figure 4.4. Figure 4.5 shows the trade-off between smoothness loss (blue line, measured by first order difference in Proposition 4.5.8) and imitation loss (red line, measured by mean squared distance) for varying σ. We navigate this trade-off by setting σ closer to 1 in early iterations, and have $\sigma \to 0$ as n increases. This "gradual increase" produces more stable policies, especially during early iterations where the learning policy tends to perform poorly (as formalized in Proposition 4.5.8). In Figure 4.4, when the initial policy (green trajectory) has poor performance, setting smooth targets (Figure 4.4b) allows learning a smooth policy in the subsequent round, in contrast to more accurate but less stable performance of "difficult" targets with low σ (Figure

4.4c-d). Figure 4.6 visualizes the behavior of the the intermediate policies learned by SIMILE, where we can see that each intermediate policy is a smooth policy.

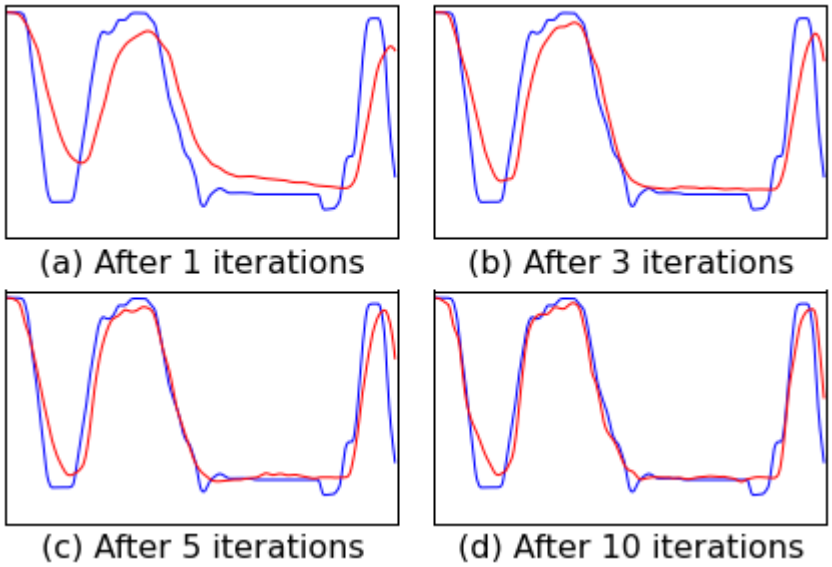


Figure 4.6: Performance after different number of iterations.

Deterministic vs. Stochastic Interpolation: Finally, we evaluate the benefits of using deterministic policy averaging intead of stochastically combine different policies, as done in SEARN. To control for other factors, we set β to a fixed value of 0.5, and keep the new training dataset $\mathcal{D}_n$ the same for each iteration n. The average imitation loss of stochastic policy sampling are evaluated after 50 stochastic roll-outs at each iterations. This average stochastic policy error tends to be higher compared to the empirical error of the deterministic trajectory, as seen from Figure 4.7, and confirms our finding from Corollary 4.5.3.

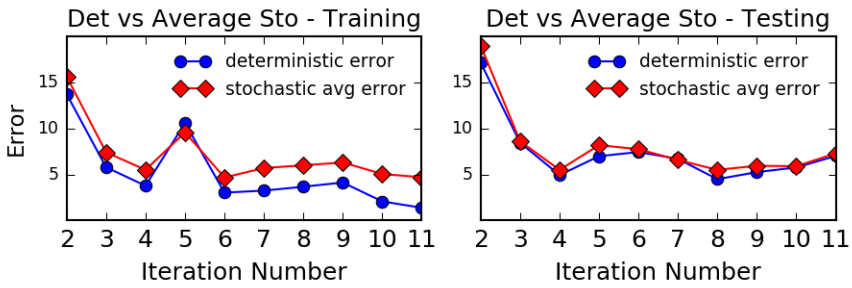


Figure 4.7: Deterministic policy error vs. average stochastic policy error for $\beta = 0.5$ and 50 roll-outs of the stochastic policies.

4.7 Discussion

We formalized the problem of smooth imitation learning for online sequence prediction, which is a variant of imitation learning that uses a notion of a smooth policy class. We proposed SIMILE (**S**mooth **IMI**tation **LE**arning), which is an iterative learning reduction approach to learning smooth policies from expert demonstrations in a continuous and dynamic environment. SIMILE utilizes an adaptive learning rate that provably allows much faster convergence compared to previous learning reduction approaches, and also enjoys better sample complexity than previous work by being fully deterministic and allowing for virtual simulation of training labels. We validated the efficiency and practicality of our approach on a setting of online camera planning.

Chapter 5

REDUCTION TO ONLINE LEARNING WITH POLICY-BASED STRUCTURE (PROGRAMMATIC REINFORCEMENT LEARNING)

Summary. We study the problem of programmatic reinforcement learning, in which policies are represented as short programs in a symbolic language. Programmatic policies can be more interpretable, generalizable, and amenable to formal verification than neural policies; however, designing rigorous learning approaches for such policies remains a challenge. Our approach to this challenge — a meta-algorithm called PROPEL— is based on three insights. First, we view our learning task as optimization in policy space, modulo the constraint that the desired policy has a programmatic representation, and solve this optimization problem using a form of mirror descent that takes a gradient step into the unconstrained policy space and then projects back onto the constrained space. Second, we view the unconstrained policy space as mixing neural and programmatic representations, which enables employing state-of-the-art deep policy gradient approaches. Third, we cast the projection step as program synthesis via imitation learning, and exploit contemporary combinatorial methods for this task. We present theoretical convergence results for PROPEL and empirically evaluate the approach in three continuous control domains. The experiments show that PROPEL can significantly outperform state-of-the-art approaches for learning programmatic policies.

5.1 Introduction to Programmatic Reinforcement Learning

A growing body of work (Verma et al., 2018; Bastani et al., 2018; Zhu et al., 2019) investigates reinforcement learning (RL) approaches that represent policies as programs in a symbolic language, e.g., a domain-specific language for composing control modules such as PID controllers (Ang et al., 2005). Short programmatic policies offer many advantages over neural policies discovered through deep RL, including greater interpretability, better generalization to unseen environments, and greater amenability to formal verification. These benefits motivate developing effective approaches for learning such programmatic policies.

However, programmatic reinforcement learning (PRL) remains a challenging problem, owing to the highly structured nature of the policy space. Recent state-of-the-art

approaches employ program synthesis methods to imitate or distill a pre-trained neural policy into short programs (Verma et al., 2018; Bastani et al., 2018). However, such a distillation process can yield a highly suboptimal programmatic policy — i.e., a large distillation gap — and the issue of direct policy search for programmatic policies also remains open.

In this chapter, we develop PROPEL (Imitation-**P**rojected **P**rogrammatic R**e**inforcement **L**earning), a new learning meta-algorithm for PRL, as a response to this challenge. The design of PROPEL is based on three insights that enables integrating and building upon state-of-the-art approaches for policy gradients and program synthesis. First, we view programmatic policy learning as a constrained policy optimization problem, in which the desired policies are constrained to be those that have a programmatic representation. This insight motivates utilizing constrained mirror descent approaches, which take a gradient step into the unconstrained policy space and then project back onto the constrained space. Second, by allowing the unconstrained policy space to have a mix of neural and programmatic representations, we can employ well-developed deep policy gradient approaches (Sutton et al., 2000; Lillicrap et al., 2015; Schulman et al., 2015, 2017; Cheng et al., 2019e) to compute the unconstrained gradient step. Third, we define the projection operator using program synthesis via imitation learning (Verma et al., 2018; Bastani et al., 2018), in order to recover a programmatic policy from the unconstrained policy space. Our contributions can be summarized as:

- We present PROPEL, a novel meta-algorithm that is based on mirror descent, program synthesis, and imitation learning, for PRL.
- On the theoretical side, we show how to cast PROPEL as a form of constrained mirror descent. We provide a thorough theoretical analysis characterizing the impact of approximate gradients and projections. Further, we prove results that provide expected regret bounds and finite-sample guarantees under reasonable assumptions.
- On the practical side, we provide a concrete instantiation of PROPEL and evaluate it in three continuous control domains, including the challenging car-racing domain TORCS (Wymann et al., 2014). The experiments show significant improvements over state-of-the-art approaches for learning programmatic policies.

5.2 Policy Learning Problem within the Structured Policy Class

The problem of programmatic reinforcement learning (PRL) consists of a Markov Decision Process (MDP) $\mathcal{M}$ and a programmatic policy class Π. The definition of $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, c, p_0, \gamma)$ is standard (Sutton and Barto, 2018b), with $\mathcal{S}$ being the state space, $\mathcal{A}$ the action space, $P(s'|s,a)$ the probability density function of transitioning from a state-action pair to a new state, $c(s,a)$ the state-action cost function, $p_0(s)$ a distribution over starting states, and $\gamma \in (0,1)$ the discount factor. A policy $\pi : \mathcal{S} \to \mathcal{A}$ (stochastically) maps states to actions. We focus on continuous control problems, so $\mathcal{S}$ and $\mathcal{A}$ are assumed to be continuous spaces. The goal is to find a programmatic policy $\pi^* \in \Pi$ such that:

$$\pi^* = \arg\min_{\pi \in \Pi} J(\pi), \qquad \text{where: } J(\pi) = \mathbf{E}\left[\sum_{i=0}^{\infty} \gamma^i c(s_i, a_i \equiv \pi(s_i))\right], \tag{5.1}$$

with the expectation taken over the initial state distribution $s_0 \sim p_0$, the policy decisions, and the transition dynamics P. One can also use rewards, in which case (5.1) becomes a maximization problem.

Programmatic Policy Class. A programmatic policy class Π consists of policies that can be represented parsimoniously by a (domain-specific) programming language. Recent work (Verma et al., 2018; Bastani et al., 2018; Zhu et al., 2019) indicates that such policies can be easier to interpret and formally verify than neural policies, and can also be more robust to changes in the environment.

In this chapter, we consider two concrete classes of programmatic policies. The first, a simplification of the class considered in Verma et al. (Verma et al., 2018), is defined by the modular, high-level language in Figure 5.1. This language assumes a library of parameterized functions $\oplus_\theta$ representing standard controllers, for instance Proportional-Integral-Derivative (PID) (Åström and Hägglund, 1984) or bang-bang controllers (Bellman et al., 1956). Programs in the language take states s as inputs and produce actions a as output, and can invoke fully instantiated library controllers along with predefined arithmetic, boolean and relational operators. The second, "lower-level" class, from Bastani et al. (Bastani et al., 2018), consists of decision trees that map states to actions.

Example. Consider the problem of learning a programmatic policy, in the language of Figure 5.1, that controls a car's accelerator in the TORCS car-racing environment (Wymann et al., 2014). Figure 5.2 shows a program in our language for this task. The program invokes PID controllers $\texttt{PID}_{\langle j,\theta_P,\theta_I,\theta_D\rangle}$, where j identifies the

$$\begin{aligned} \pi(s) &::= a \mid Op(\pi_1(s), \ldots, \pi_k(s)) \mid \textbf{if } b \textbf{ then } \pi_1(s) \textbf{ else } \pi_2(s) \mid \oplus_\theta(\pi_1(s), \ldots, \pi_k(s)) \\ b &::= \phi(s) \mid BOp(b_1, \ldots, b_k) \end{aligned}$$

Figure 5.1: *A high-level syntax for programmatic policies, inspired by (Verma et al., 2018). A policy $\pi(s)$ takes a state s as input and produces an action a as output. b represents boolean expressions; ϕ is a boolean-valued operator on states; Op is an operator that combines multiple policies into one policy; BOp is a standard boolean operator; and $\oplus_\theta$ is a "library function" parameterized by θ.*

$$\begin{aligned} &\textbf{if } (s[\texttt{TrackPos}] < 0.011 \textbf{ and } s[\texttt{TrackPos}] > -0.011) \\ &\qquad \textbf{then } \texttt{PID}_{\langle \texttt{RPM},0.45,3.54,0.03,53.39 \rangle}(s) \textbf{ else } \texttt{PID}_{\langle \texttt{RPM},0.39,3.54,0.03,53.39 \rangle}(s) \end{aligned}$$

Figure 5.2: *A programmatic policy for acceleration in TORCS (Wymann et al., 2014), automatically discovered by PROPEL.* $s[\texttt{TrackPos}]$ *represents the most recent reading from sensor* `TrackPos`.

sensor (out of 29, in our experiments) that provides inputs to the controller, and θ_P, θ_I, and θ_D are respectively the real-valued coefficients of the proportional, integral, and derivative terms in the controller. We note that the program only uses the sensors `TrackPos` and `RPM`. While `TrackPos` (for the position of the car relative to the track axis) is used to decide which controller to use, only the `RPM` sensor is needed to calculate the acceleration.

Learning Challenges. Learning programmatic policies in the continuous RL setting is challenging, as the best performing methods utilize policy gradient approaches (Sutton et al., 2000; Lillicrap et al., 2015; Schulman et al., 2015, 2017; Cheng et al., 2019e), but policy gradients are hard to compute in programmatic representations. In many cases, Π may not even be differentiable. For our approach, we only assume access to program synthesis methods that can select a programmatic policy $\pi \in \Pi$ that minimizes imitation disagreement with demonstrations provided by a teaching oracle. Because imitation learning tends to be easier than general RL in long-horizon tasks (Sun et al., 2017), the task of imitating a neural policy with a program is, intuitively, significantly simpler than the full programmatic RL problem. This intuition is corroborated by past work on programmatic RL (Verma et al., 2018), which shows that direct search over programs often fails to meet basic performance objectives.

Algorithm 5 Imitation-Projected Programmatic Reinforcement Learning (PROPEL)

Input: Programmatic & Neural Policy Classes: Π & $\mathcal{F}$.
Input: Either initial π_0 or initial f_0
Define joint policy class: $\mathcal{H} \equiv \Pi \oplus \mathcal{F}$ *//$h \equiv \pi + f$ defined as $h(s) = \pi(s) + f(s)$*
if given initial f_0 **then**
  $\pi_0 \leftarrow \textsc{Project}(f_0)$ *//program synthesis via imitation learning*
end if
for $t = 1, \ldots, T$ **do**
  $h_t \leftarrow \textsc{Update}_{\mathcal{F}}(\pi_{t-1}, \eta)$ *//policy gradient in neural policy space with learning rate η*
  $\pi_t \leftarrow \textsc{Project}_{\Pi}(h_t)$ *//program synthesis via imitation learning*
end for
Return: Policy π_T

5.3 Learning Algorithm via Reduction to Online Learning

To develop our approach, we take the viewpoint of (5.1) being a constrained optimization problem, where $\Pi \subset \mathcal{H}$ resides within a larger space of policies $\mathcal{H}$. In particular, we will represent $\mathcal{H} \equiv \Pi \oplus \mathcal{F}$ using a mixing of programmatic policies Π and neural polices $\mathcal{F}$. Any mixed policy $h \equiv \pi + f$ can be invoked as $h(s) = \pi(s) + f(s)$. In general, we assume that $\mathcal{F}$ is a good approximation of Π (i.e., for each $\pi \in \Pi$ there is some $f \in \mathcal{F}$ that approximates it well), which we formalize in Section 5.4.

We can now frame our constrained learning problem as minimizing (5.1) over $\Pi \subset \mathcal{H}$, that alternate between taking a gradient step in the general space $\mathcal{H}$ and projecting back down onto Π. This "lift-and-project" perspective motivates viewing our problem via the lens of mirror descent (Nemirovsky and Yudin, 1983). In standard mirror descent, the unconstrained gradient step can be written as $h \leftarrow h_{prev} - \eta \nabla_{\mathcal{H}} J(h_{prev})$ for step size η, and the projection can be written as $\pi \leftarrow \arg\min_{\pi' \in \Pi} D(\pi', h)$ for divergence measure D.

Our approach, *Imitation-Projected Programmatic Reinforcement Learning* (PROPEL), is outlined in Algorithm 5 (also see Figure 5.3). PROPEL is a meta-algorithm that requires instantiating two subroutines, UPDATE and PROJECT, which correspond to the standard update and projection steps, respectively. PROPEL can be viewed as a form of functional mirror descent with some notable deviations from vanilla mirror descent.

UPDATE$_{\mathcal{F}}$. Since policy gradient methods are well-developed for neural policy classes $\mathcal{F}$ (e.g., (Lillicrap et al., 2015; Schulman et al., 2015, 2017; Henderson et al., 2018; Duan et al., 2016; Cheng et al., 2019e)) and non-existent for programmatic

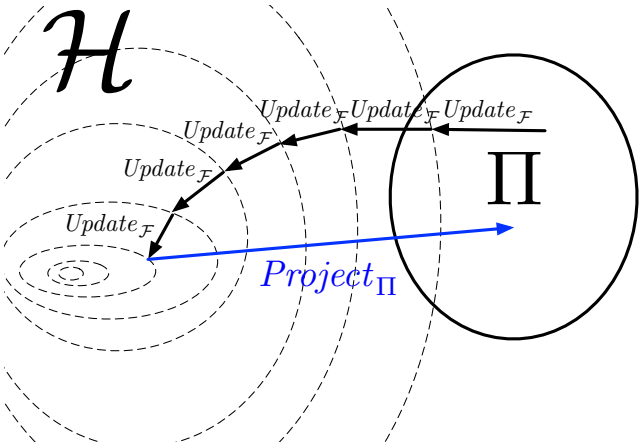


Figure 5.3: Depicting the PROPEL meta-algorithm.

policy classes Π, PROPEL is designed to leverage policy gradients in $\mathcal{F}$ and avoid policy gradients in Π. Algorithm 6 shows one instantiation of $\text{UPDATE}_{\mathcal{F}}$. Note that standard mirror descent takes unconstrained gradient steps in $\mathcal{H}$ rather than $\mathcal{F}$, and we discuss this discrepancy between $\text{UPDATE}_{\mathcal{H}}$ and $\text{UPDATE}_{\mathcal{F}}$ in Section 5.4.

PROJECT_{Π}. Projecting onto Π can be implemented using program synthesis via imitation learning, i.e., by synthesizing a $\pi \in \Pi$ to best imitate demonstrations provided by a teaching oracle $h \in \mathcal{H}$. Recent work (Verma et al., 2018; Bastani et al., 2018; Zhu et al., 2019) has given practical heuristics for this task for various programmatic policy classes. Algorithm 7 shows one instantiation of PROJECT_{Π} (based on DAgger (Ross et al., 2011a)). One complication that arises is that finite-sample runs of such imitation learning approaches only return approximate solutions and so the projection is not exact. We characterize the impact of approximate projections in Section 5.4.

Practical Considerations. In practice, we often employ multiple gradient steps before taking a projection step (as also described in Algorithm 6), because the step size of individual (stochastic) gradient updates can be quite small. Another issue that arises in virtually all policy gradient approaches is that the gradient estimates can have very high variance (Sutton et al., 2000; Konda and Tsitsiklis, 2000; Henderson et al., 2018). We utilize low-variance policy gradient updates by using the reference π as a proximal regularizer in function space (Cheng et al., 2019e).

For the projection step (Algorithm 7), in practice we often retain all previous roll-outs τ from all previous projection steps. It is straightforward to query the current oracle h to provide demonstrations on the states $s \in \tau$ from previous roll-outs, which can lead to substantial savings in sample complexity with regards to executing roll-outs on the environment, while not harming convergence.

Algorithm 6 UPDATE$_{\mathcal{F}}$: neural policy gradient for mixed policies

Input: Neural Policy Class $\mathcal{F}$. **Input:** Reference programmatic policy: π
Input: Step size: η. **Input:** Regularization parameter: λ
Initialize neural policy: f_0 *//any standard randomized initialization*
for $j = 1, \ldots, m$ **do**
  $f_j \leftarrow f_{j-1} - \eta\lambda\nabla_{\mathcal{F}} J(\pi + \lambda f_{j-1})$ *//using DDPG (Lillicrap et al., 2015), TRPO (Schulman et al., 2015), etc., holding π fixed*
end for
Return: $h \equiv \pi + \lambda f_m$

Algorithm 7 PROJECT$_{\Pi}$: program synthesis via imitation learning

Input: Programmatic Policy Class: Π. **Input:** Oracle policy: h
Roll-out h on environment, get trajectory: $\tau_0 = (s^0, h(s^0), s^1, h(s^1), \ldots)$
Create supervised demonstration set: $\Gamma_0 = \{(s, h(s))\}$ from τ_0
Derive π_0 from Γ_0 via program synthesis *//e.g., using methods in (Verma et al., 2018; Bastani et al., 2018)*
for $k = 1, \ldots, M$ **do**
  Roll-out π_{k-1}, creating trajectory: τ_k
  Collect demonstration data: $\Gamma' = \{(s, h(s)) | s \in \tau_k\}$
  $\Gamma_k \leftarrow \Gamma' \cup \Gamma_{k-1}$ *//DAgger-style imitation learning (Ross et al., 2011a)*
  Derive π_k from Γ_k via program synthesis *//e.g., using methods in (Verma et al., 2018; Bastani et al., 2018)*
end for
Return: π_M

5.4 Theoretical Analysis

We start by viewing PROPEL through the lens of online learning in function space, independent of the specific parametric representation. This start point yields a convergence analysis of Alg. 5 in Section 5.4 under generic approximation errors. We then analyze the issues of policy class representation in Sections 5.4 and 5.4, and connect Algorithms 6 and 7 with the overall performance, under some simplifying conditions. In particular, Section 5.4 characterizes the update error in a possibly non-differentiable setting; to our knowledge, this is the first such analysis of its kind for reinforcement learning.

Preliminaries. We consider Π and $\mathcal{F}$ to be subspaces of an ambient policy space $\mathcal{U}$, which is a vector space equipped with inner product $\langle \cdot, \cdot \rangle$, induced norm $\|u\| = \sqrt{\langle u, u \rangle}$, dual norm $\|v\|_* = \sup\{\langle v, u \rangle | \, \|u\| \leqslant 1\}$, and standard scaling & addition: $(au + bv)(s) = au(s) + bv(s)$ for $a, b \in \mathbb{R}$ and $u, v \in \mathcal{U}$. The cost functional of a policy u is $J(u) = \int_{\mathcal{S}} c(s, u(s)) d\mu^u(s)$, where μ^u is the distribution of states induced by u. The joint policy class is $\mathcal{H} = \Pi \oplus \mathcal{F}$, by $\mathcal{H} = \{\pi + f | \forall \pi \in \Pi, f \in \mathcal{F}\}$.[1] Note that $\mathcal{H}$ is a subspace of $\mathcal{U}$, and inherits its vector space properties. Without

[1] The operator $\oplus$ is not a direct sum, since Π and $\mathcal{F}$ are not orthogonal.

affecting the analysis, we simply equate $\mathcal{U} \equiv \mathcal{H}$ for the remainder of the chapter.

We assume that J is convex in $\mathcal{H}$, which implies that subgradient $\partial J(h)$ exists (with respect to $\mathcal{H}$) (Bauschke et al., 2011). Where J is differentiable, we utilize the notion of a Fréchet gradient. Recall that a bounded linear operator $\nabla : \mathcal{H} \mapsto \mathcal{H}$ is called a Fréchet functional gradient of J at $h \in \mathcal{H}$ if $\lim_{\|g\|\to 0} \frac{J(h+g)-J(h)-\langle \nabla J(h), g\rangle}{\|g\|} = 0$. By default, ∇ (or $\nabla_{\mathcal{H}}$ for emphasis) denotes the gradient with respect to $\mathcal{H}$, whereas $\nabla_{\mathcal{F}}$ defines the gradient in the restricted subspace $\mathcal{F}$.

PROPEL as (Approximate) Functional Mirror Descent

For our analysis, PROPEL can be viewed as approximating mirror descent in (infinite-dimensional) function space over a convex set $\Pi \subset \mathcal{H}$.[2] Similar to the finite-dimensional setting (Nemirovsky and Yudin, 1983), we choose a strongly convex and smooth functional regularizer R to be the mirror map. From the approximate mirror descent perspective, for each iteration t:

1. Obtain a noisy gradient estimate: $\widehat{\nabla}_{t-1} \approx \nabla J(\pi_{t-1})$
2. $\text{UPDATE}_{\mathcal{H}}(\pi)$ in $\mathcal{H}$ space: $\nabla R(h_t) = \nabla R(\pi_{t-1}) - \eta \widehat{\nabla}_{t-1}$ *(Note $\text{UPDATE}_{\mathcal{H}} \neq \text{UPDATE}_{\mathcal{F}}$)*
3. Obtain approximate projection: $\pi_t = \text{PROJECT}^R_{\Pi}(h_t) \approx \arg\min_{\pi\in\Pi} D_R(\pi, h_t)$

$D_R(u,v) = R(u) - R(v) - \langle \nabla R(u), u - v\rangle$ is a Bregman divergence. Taking $R(h) = \frac{1}{2}\|h\|^2$ will recover projected functional gradient descent in $L2$-space. Here UPDATE becomes $h_t = \pi_{t-1} - \eta\widehat{\nabla} J(\pi_{t-1})$, and PROJECT solves for $\arg\min_{\pi\in\Pi} \|\pi - h_t\|^2$. While we mainly focus on this choice of R in our experiments, note that other selections of R lead to different UPDATE and PROJECT operators (e.g., minimizing KL divergence if R is negative entropy).

The functional mirror descent scheme above may encounter two additional sources of error compared to standard mirror descent (Nemirovsky and Yudin, 1983). First, in the stochastic setting (also called bandit feedback (Flaxman et al., 2005)), the gradient estimate $\widehat{\nabla}_t$ may be biased, in addition to having high variance. One potential source of bias is the gap between $\text{UPDATE}_{\mathcal{H}}$ and $\text{UPDATE}_{\mathcal{F}}$. Second, the PROJECT step may be inexact. We start by analyzing the behavior of PROPEL under generic bias, variance, and projection errors, before discussing the implications of approximating $\text{UPDATE}_{\mathcal{H}}$ and PROJECT_{Π} by Algs. 6 & 7, respectively. Let the bias

[2] Π can be convexified by considering *randomized* policies, as stochastic combinations of $\pi \in \Pi$ (cf. (Le et al., 2019b)).

be bounded by β, i.e., $\left\|\mathbb{E}[\widehat{\nabla}_t|\pi_t] - \nabla J(\pi_t)\right\|_* \leqslant \beta$ almost surely. Similarly let the variance of the gradient estimate be bounded by σ^2, and the projection error norm $\|\pi_t - \pi_t^*\| \leqslant \epsilon$. We state the expected regret bound below; more details and a proof appear in Appendix D.1.

Theorem 5.4.1 (Expected regret bound under gradient estimation and projection errors)**.** *Let $\pi_1, \ldots, \pi_T$ be a sequence of programmatic policies returned by Algorithm 5, and π^* be the optimal programmatic policy. Choosing learning rate $\eta = \sqrt{\frac{1}{\sigma^2}(\frac{1}{T} + \epsilon)}$, we have the expected regret over T iterations:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon} + \beta\right). \tag{5.2}$$

The result shows that error ϵ from PROJECT and the bias β do not accumulate and simply contribute an additive term on the expected regret.[3] The effect of variance of gradient estimate decreases at a $\sqrt{1/T}$ rate. Note that this regret bound is agnostic to the specific UPDATE and PROJECT operations, and can be applied more generically beyond the specific algorithmic choices used in this section of the thesis.

Finite-Sample Analysis under Vanilla Policy Gradient Update and DAgger Projection

Next, we show how certain instantiations of UPDATE and PROJECT affect the magnitude of errors and influence end-to-end learning performance from finite samples, under some simplifying assumptions on the UPDATE step. For this analysis, we simplify Alg. 6 into the case $\text{UPDATE}_{\mathcal{F}} \equiv \text{UPDATE}_{\mathcal{H}}$. In particular, we assume programmatic policies in Π to be parameterized by a vector $\theta \in \mathbb{R}^k$, and π is differentiable in θ (e.g., we can view $\Pi \subset \mathcal{F}$ where $\mathcal{F}$ is parameterized in $\mathbb{R}^k$). We further assume the trajectory roll-out is performed in an exploratory manner, where action is taken uniformly random over finite set of A actions, thus enabling the bound on the bias of gradient estimates via Bernstein's inequality. The PROJECT step is consistent with Alg. 7, i.e., using DAgger (Ross et al., 2011b) under convex imitation loss, such as ℓ_2 loss. We have the following high-probability guarantee:

Theorem 5.4.2 (Finite-sample guarantee)**.** *At each iteration, we perform vanilla policy gradient estimate of π (over $\mathcal{H}$) using m trajectories and, use DAgger algorithm to collect M roll-outs for the imitation learning projection. Setting the*

[3] Other mirror descent-style analyses, such as in (Sun et al., 2018b), lead to accumulation of errors over the rounds of learning T. One key difference is that we are leveraging the assumption of convexity of J in the (infinite-dimensional) function space representation.

learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}\right)}$*, after* T *rounds of the algorithm, we have that:*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leqslant$$
$$O\left(\sigma\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right) + O\left(\sigma\sqrt{\frac{\log(Tk/\delta)}{m}} + \frac{AH\log(Tk/\delta)}{m}\right)$$

holds with probability at least $1-\delta$*, with* H *being the task horizon,* A *the cardinality of action space,* σ^2 *the variance of policy gradient estimates, and* k *the dimension* Π*'s parameterization.*

The expanded result and proof are included in Appendix D.1. The proof leverages previous analysis from DAgger (Ross et al., 2011a) and the finite sample analysis of vanilla policy gradient algorithm (Kakade et al., 2003). The finite-sample regret bound scales linearly with the standard deviation σ of the gradient estimate, while the bias, which is the very last component of the RHS, scales linearly with the task horizon H. Note that the standard deviation σ can be exponential in task horizon H in the worst case (Kakade et al., 2003), and so it is important to have practical implementation strategies to reduce the variance of the UPDATE operation. While conducted in a stylized setting, this analysis provides insight in the relative trade-offs of spending effort in obtaining more accurate projections versus more reliable gradient estimates.

Closing the gap between UPDATE$_{\mathcal{H}}$ and UPDATE$_{\mathcal{F}}$

Our functional mirror descent analysis rests on taking gradients in $\mathcal{H}$: UPDATE$_{\mathcal{H}}(\pi)$ involves estimating $\nabla_{\mathcal{H}} J(\pi)$ in the $\mathcal{H}$ space. On the other hand, Algorithm 6 performs UPDATE$_{\mathcal{F}}(\pi)$ only in the neural policy space $\mathcal{F}$. In either case, although $J(\pi)$ may be differentiable in the non-parametric ambient policy space, it may not be possible to obtain a differentiable parametric programmatic representation in Π. In this section, we discuss theoretical motivations to addressing a practical issue: *How do we define and approximate the gradient* $\nabla_{\mathcal{H}} J(\pi)$ *under a parametric representation?* To our knowledge, we are the first to consider such a theoretical question for reinforcement learning.

Defining a consistent approximation of $\nabla_{\mathcal{H}} J(\pi)$. The idea in UPDATE$_{\mathcal{F}}(\pi)$ (Line 8 of Alg. 5) is to approximate $\nabla_{\mathcal{H}} J(\pi)$ by $\nabla_{\mathcal{F}} J(f)$, which has a differentiable

representation, at some f close to π (under the norm). Under appropriate conditions on $\mathcal{F}$, we show that this approximation is valid.

Proposition 5.4.3. *Assume that (i) J is Fréchet differentiable on $\mathcal{H}$, (ii) J is also differentiable on the restricted subspace $\mathcal{F}$, and (iii) $\mathcal{F}$ is dense in $\mathcal{H}$ (i.e., the closure $\bar{\mathcal{F}} = \mathcal{H}$). Then for any fixed policy $\pi \in \Pi$, define a sequence of policies $f_k \in \mathcal{F}$, $k = 1, 2, \ldots$), that converges to π:* $\lim_{k\to\infty} \|f_k - \pi\| = 0$. *We then have* $\lim_{k\to\infty} \|\nabla_{\mathcal{F}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$.

Since the Fréchet gradient is unique in the ambient space $\mathcal{H}$, $\forall k$ we have $\nabla_{\mathcal{H}} J(f_k) = \nabla_{\mathcal{F}} J(f_k) \to \nabla_{\mathcal{H}} J(\pi)$ as $k \to \infty$ (by Proposition 5.4.3). We thus have an asymptotically unbiased approximation of $\nabla_{\mathcal{H}} J(\pi)$ via differentiable space $\mathcal{F}$ as: $\nabla_{\mathcal{F}} J(\pi) \triangleq \nabla_{\mathcal{H}} J(\pi) \triangleq \lim_{k\to\infty} \nabla_{\mathcal{F}} J(f_k)$.[4] Connecting to the result from Theorem 5.4.1, let σ^2 be an upper bound on the policy gradient estimates in the *neural policy class* $\mathcal{F}$, under an asymptotically unbiased approximation of $\nabla_{\mathcal{H}} J(\pi)$, the expected regret bound becomes $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon}\right)$.

Bias-variance considerations of UPDATE$_{\mathcal{F}}(\pi)$ To further theoretically motivate a practical strategy for UPDATE$_{\mathcal{F}}(\pi)$ in Algorithm 6, we utilize an equivalent proximal perspective of mirror descent (Beck and Teboulle, 2003), where UPDATE$_{\mathcal{H}}(\pi)$ is equivalent to solving for $h' = \arg\min_{h\in\mathcal{H}} \eta\langle \nabla_{\mathcal{H}} J(\pi), h\rangle + D_R(h, \pi)$.

Proposition 5.4.4 (Minimizing a relaxed objective)**.** *For a fixed programmatic policy π, with sufficiently small constant $\lambda \in (0, 1)$, we have that*

$$\min_{h\in\mathcal{H}} \eta\langle \nabla_{\mathcal{H}} J(\pi), h)\rangle + D_R(h, \pi) \leqslant \min_{f\in\mathcal{F}} J\big(\pi + \lambda f\big) - J(\pi) + \langle \nabla J(\pi), \pi\rangle \quad (5.3)$$

Thus, a relaxed UPDATE$_{\mathcal{H}}$ step is obtained by minimizing the RHS of (5.3), i.e., minimizing $J(\pi + \lambda f)$ over $f \in \mathcal{F}$. Each gradient descent update step is now $f' = f - \eta\lambda\nabla_{\mathcal{F}} J(\pi_t + \lambda f)$, corresponding to Line 5 of Algorithm 6. For fixed π and small λ, this relaxed optimization problem becomes regularized policy optimization over $\mathcal{F}$, which is significantly easier. Functional regularization in policy space around a fixed prior controller π has demonstrated significant reduction in the variance of gradient estimate (Cheng et al., 2019e), at the expense of some bias. The below expected regret bound summarizes the impact of this increased bias and reduced variance, with details included in Appendix D.1.

[4] We do not assume $J(\pi)$ to be differentiable when restricting to the policy subspace Π, i.e., $\nabla_{\Pi} J(\pi)$ may not exist under policy parameterization of Π.

Proposition 5.4.5 (Bias-variance characterization of UPDATE$_{\mathcal{F}}$)**.** *Assuming $J(h)$ is L-strongly smooth over $\mathcal{H}$, i.e., $\nabla_{\mathcal{H}} J(h)$ is L-Lipschitz continuous, approximating UPDATE$_{\mathcal{H}}$ by UPDATE$_F$ per Alg. 6 leads to the expected regret bound:* $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\lambda\sigma\sqrt{\frac{1}{T}+\epsilon} + \lambda^2 L^2\right)$.

Compared to the idealized unbiased approximation in Proposition 5.4.3, the introduced bias here is related to the inherent smoothness property of cost functional $J(h)$ over the joint policy class $\mathcal{H}$, i.e., how close $J(\pi + \lambda f)$ is to its linear underapproximation $J(\pi) + \langle \nabla_{\mathcal{H}} J(\pi), \lambda f \rangle$ around π.

5.5 Experiments

We demonstrate the effectiveness of PROPEL in synthesizing programmatic controllers in three continuous control environments. For brevity and focus, this section primarily focuses on TORCS[5], a challenging race car simulator environment (Wymann et al., 2014). Empirical results on two additional classic control tasks, Mountain-Car and Pendulum, are provided in Appendix D.2; those results follow similar trends as the ones described for TORCS below, and further validate the convergence analysis of PROPEL.

Experimental Setup. We evaluate over five distinct tracks in the TORCS simulator. The difficulty of a track can be characterized by three properties; track length, track width, and number of turns. Our suite of tracks provides environments with varying levels of difficulty for the learning algorithm. The performance of a policy in the TORCS simulator is measured by the *lap time* achieved on the track. To calculate the lap time, the policies are allowed to complete a three-lap race, and we record the best lap time during this race. We perform the experiments with twenty-five random seeds and report the median lap time over these twenty-five trials. Some of the policies crash the car before completing a lap on certain tracks, even after training for 600 episodes. Such crashes are recorded as a lap time of infinity while calculating the median. If the policy crashes for more than half the seeds, this is reported as CR in Tables 5.1 & 5.2. We choose to report the median because taking the crash timing as infinity, or an arbitrarily large constant, heavily skews other common measures such as the mean.

Baselines. Among recent state-of-the-art approaches to learning programmatic policies are NDPS (Verma et al., 2018) for high-level language policies, and VIPER

[5]The code for the TORCS experiments can be found at: https://bitbucket.org/averma8053/propel

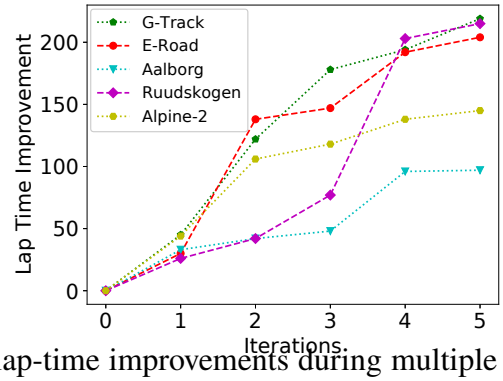


Figure 5.4: Median lap-time improvements during multiple iterations of IPPGPROGRAM over 25 randoı

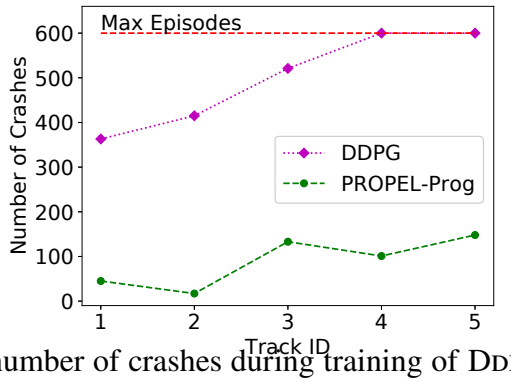


Figure 5.5: Median number of crashes during training of DDPG and IPPGPROGRAM over 25 random seeds.

(Bastani et al., 2018) for learning tree-based policies. Both NDPS and VIPER rely on imitating a fixed (pre-trained) neural policy oracle, and can be viewed as degenerate versions of PROPEL that only run Lines 4-6 in Algorithm 5. We present two PROPEL analogues to NDPS and VIPER: (i) IPPGPROGRAM: PROPEL using the high-level language of Figure 5.1 as the class of programmatic policies, similar to NDPS. (ii) IPPGTREE: PROPEL using regression trees, similar to VIPER. We also report results for PRIOR, which is a (sub-optimal) PID controller that is also used as the initial policy in PROPEL. In addition, to study generalization ability as well as safety behavior during training, we also include DDPG, a neural policy learned using the Deep Deterministic Policy Gradients (Lillicrap et al., 2015) algorithm, with 600 episodes of training for each track. In principle, PROPEL and its analysis can accommodate different policy gradient subroutines. However, in the TORCS domain, other policy gradient algorithms such as PPO and TRPO failed to learn policies that are able to complete the considered tracks. We thus focus on DDPG as our main policy gradient component.

Evaluating Performance. Table 5.1 shows the performance on the considered TORCS tracks. We see that IPPGPROGRAM and IPPGTREE consistently outperform the NDPS (Verma et al., 2018) and VIPER (Bastani et al., 2018) baselines, respectively.

Table 5.1: *Performance results in TORCS over 25 random seeds. Each entry is formatted as Lap-time / Crash-ratio, reporting median lap time in seconds over all the seeds (lower is better) and ratio of seeds that result in crashes (lower is better). A lap time of CR indicates the agent crashed and could not complete a lap for more than half the seeds.*

LENGTH	G-TRACK 3186M	E-ROAD 3260M	AALBORG 2588M	RUUDSKOGEN 3274M	ALPINE-2 3774M
PRIOR	312.92 / 0.0	322.59 / 0.0	244.19 / 0.0	340.29 / 0.0	402.89 / 0.0
DDPG	78.82 / 0.24	89.71 / 0.28	101.06 / 0.40	CR / 0.68	CR / 0.92
NDPS	108.25 / 0.24	126.80 / 0.28	163.25 / 0.40	CR / 0.68	CR / 0.92
VIPER	83.60 / 0.24	87.53 / 0.28	110.57 / 0.40	CR / 0.68	CR / 0.92
IPPGPROGRAM	93.67 / 0.04	119.17 / 0.04	147.28 / 0.12	124.58 / 0.16	256.59 / 0.16
IPPGTREE	78.33 / 0.04	79.39 / 0.04	109.83 / 0.16	118.80 / 0.24	236.01 / 0.36

While DDPG outperforms PROPEL on some tracks, its volatility causes it to be unable to learn in some environments, and hence to crash the majority of the time. Figure 5.4 shows the consistent improvements made over the prior by IPPGPROGRAM, over the iterations of the PROPEL algorithm. Appendix D.2 contains similar results achieved on the two classic control tasks, MountainCar and Pendulum. Figure 5.5 shows that, compared to DDPG, our approach suffers far fewer crashes while training in TORCS.

Evaluating Generalization. To compare the ability of the controllers to perform on tracks not seen during training, we executed the learned policies on all the other tracks (Table 5.2). We observe that DDPG crashes significantly more often than IPPGPROGRAM. This demonstrates the generalizability of the policies returned by PROPEL. Generalization results for the IPPGTREE policy are given in the appendix. In general, IPPGTREE policies are more generalizable than DDPG but less than IPPGPROGRAM. On an absolute level, the generalization ability of PROPEL still leaves much room for improvement, which is an interesting direction for future work.

Verifiability of Policies. As shown in prior work (Bastani et al., 2018; Verma et al., 2018), parsimonious programmatic policies are more amenable to formal verification than neural policies. Unsurprisingly, the policies generated by IPPGTREE and IPPGPROGRAM are easier to verify than DDPG policies. As a concrete example, we verified a smoothness property of the IPPGPROGRAM policy using the Z3 SMT-solver (de Moura and Bjørner, 2008) (more details in Appendix D.2). The verification terminated in 0.49 seconds.

Initialization. In principle, PROPEL can be initialized with a random program, or a random policy trained using DDPG. In practice, the performance of PROPEL

Table 5.2: *Generalization results in TORCS, where rows are training and columns are testing tracks. Each entry is formatted as IPPGPROGRAM / DDPG, and the number reported is the median lap time in seconds over all the seeds (lower is better). CR indicates the agent crashed and could not complete a lap for more than half the seeds.*

	G-TRACK	E-ROAD	AALBORG	RUUDSKOGEN	ALPINE-2
G-TRACK	-	124 / CR	CR / CR	CR / CR	CR / CR
E-ROAD	102 / 92	-	CR / CR	CR / CR	CR / CR
AALBORG	201 / 91	228 / CR	-	217 / CR	CR / CR
RUUDSKOGEN	131 / CR	135 / CR	CR / CR	-	CR / CR
ALPINE-2	222 / CR	231 / CR	184 / CR	CR / CR	-

depends to a certain degree on the stability of the policy gradient procedure, which is DDPG in our experiments. Unfortunately, DDPG often exhibits high variance across trials and fares poorly in challenging RL domains. Specifically, in our TORCS experiments, DDPG fails on a number of tracks (similar phenomena have been reported in previous work that experiments on similar continuous control domains (Henderson et al., 2018; Cheng et al., 2019e; Verma et al., 2018)). Agents obtained by initializing PROPEL with neural policies obtained via DDPG also fail on multiple tracks. Their performance over the five tracks is reported in Appendix D.2. In contrast, PROPEL can often finish the challenging tracks when initialized with a very simple hand-crafted programmatic prior.

5.6 Related Work

Program Synthesis. Program synthesis is the problem of automatically searching for a program within a language that fits a given specification (Gulwani et al., 2017). Recent approaches to the problem have leveraged symbolic knowledge about program structure (Feser et al., 2015), satisfiability solvers (Solar-Lezama et al., 2006; Jha et al., 2010), and meta-learning techniques (Murali et al., 2018; Parisotto et al., 2016; Devlin et al., 2017; Balog et al., 2017) to generate interesting programs in many domains (Alur et al., 2013; Polozov and Gulwani, 2015; Alur et al., 2017). In most prior work, the specification is a logical constraint on the input/output behavior of the target program. However, there is also a growing body of work that considers program synthesis modulo optimality objectives (Bloem et al., 2009; Chaudhuri et al., 2014; Raychev et al., 2016), often motivated by machine learning tasks (Murali et al., 2018; Valkov et al., 2018; Ellis et al., 2018; Du et al., 2018; Verma et al., 2018; Bastani et al., 2018; Zhu et al., 2019). Synthesis of programs

that imitates an oracle has been considered in both the logical (Jha et al., 2010) and the optimization (Verma et al., 2018; Bastani et al., 2018; Zhu et al., 2019) settings. The projection step in PROPEL builds on this prior work. While our current implementation of this step is entirely symbolic, in principle, the operation can also utilize contemporary techniques for learning policies that guide the synthesis process (Murali et al., 2018; Balog et al., 2017; Si et al., 2019).

Constrained Policy Learning. Constrained policy learning has seen increased interest in recent years, largely due to the desire to impose side guarantees such as stability and safety on the policy's behavior. Broadly, there are two approaches to imposing constraints: specifying constraints as an additional cost function (Achiam et al., 2017; Le et al., 2019b), and explicitly encoding constraints into the policy class (Alshiekh et al., 2018; Le et al., 2016b; Cheng et al., 2019e; Dalal et al., 2018; Berkenkamp et al., 2017). In some cases, these two approaches can be viewed as duals of each other. For instance, recent work that uses control-theoretic policies as a functional regularizer (Le et al., 2016b; Cheng et al., 2019e) can be viewed from the perspective of both regularization (additional cost) and an explicitly constrained policy class (a specific mix of neural and control-theoretic policies). We build upon this perspective to develop the gradient update step in our approach.

RL using Imitation Learning. There are two ways to utilize imitation learning subroutines within RL. First, one can leverage limited-access or sub-optimal experts to speed up learning (Ross and Bagnell, 2014; Cheng et al., 2019a; Chang et al., 2015; Sun et al., 2018a). Second, one can learn over two policy classes (or one policy and one model class) to achieve accelerated learning compared to using only one policy class (Montgomery and Levine, 2016; Cheng et al., 2019c; Sun et al., 2018b; Cheng et al., 2019b). Our approach has some stylistic similarities to previous efforts (Montgomery and Levine, 2016; Sun et al., 2018b) that use a richer policy space to search for improvements before re-training the primary policy to imitate the richer policy. One key difference is that our primary policy is programmatic and potentially non-differentiable. A second key difference is that our theoretical framework takes a functional gradient descent perspective — it would be interesting to carefully compare with previous analysis techniques to find a unifying framework.

RL with Mirror Descent. The mirror descent framework has previously used to analyze and design RL algorithms. For example, Thomas et al. (Thomas et al., 2013) and Mahadevan and Liu (Mahadevan and Liu, 2012) use composite objective mirror descent, or COMID (Duchi et al., 2010), which allows incorporating adap-

tive regularizers into gradient updates, thus offering connections to either natural gradient RL (Thomas et al., 2013) or sparsity inducing RL algorithms (Mahadevan and Liu, 2012). Unlike the approach in this section, these prior approaches perform projection into the same native, differentiable representation. Also, the analyses in these prior work do not consider errors introduced by hybrid representations and approximate projection operators. However, one can potentially extend our approach with versions of mirror descent, e.g., COMID, that were considered in these efforts.

5.7 Conclusion and Future Work

We have presented PROPEL, a meta-algorithm based on mirror descent, program synthesis, and imitation learning, for programmatic reinforcement learning (PRL). We have presented theoretical convergence results for PROPEL, developing novel analyses to characterize approximate projections and biased gradients within the mirror descent framework. We also validated PROPEL empirically, and show that it can discover interpretable, verifiable, generalizable, performant policies and significantly outperform the state of the art in PRL.

The central idea of PROPEL is the use of imitation learning and combinatorial methods in implementing a projection operation for mirror descent, with the goal of optimization in a functional space that lacks gradients. While we have developed PROPEL in an RL setting, this idea is not restricted to RL or even sequential decision making. Future work will seek to exploit this insight in other machine learning and program synthesis settings.

Chapter 6

HIERARCHICAL IMITATION AND REINFORCEMENT LEARNING

Summary. We study how to effectively leverage expert feedback to learn sequential decision-making policies. We focus on problems with sparse rewards and long time horizons, which typically pose significant challenges in reinforcement learning. We propose an algorithmic framework, called *hierarchical guidance*, that leverages the hierarchical structure of the underlying problem to integrate different modes of expert interaction. Our framework can incorporate different combinations of imitation learning (IL) and reinforcement learning (RL) at different levels, leading to dramatic reductions in both expert effort and cost of exploration. Using long-horizon benchmarks, including Montezuma's Revenge, we demonstrate that our approach can learn significantly faster than hierarchical RL, and be significantly more label-efficient than standard IL. We also theoretically analyze labeling cost for certain instantiations of our framework.

6.1 Introduction

Learning good agent behavior from reward signals alone—the goal of reinforcement learning (RL)—is particularly difficult when the planning horizon is long and rewards are sparse. One successful method for dealing with such long horizons is imitation learning (IL) (Abbeel and Ng, 2004; Daumé III et al., 2009; Ross et al., 2011b; Ho and Ermon, 2016), in which the agent learns by watching and possibly querying an expert. One limitation of existing imitation learning approaches is that they may require a large amount of demonstration data in long-horizon problems.

The central question we address in this chapter of the thesis is: *when experts are available, how can we most effectively leverage their feedback?* A common strategy to improve sample efficiency in RL over long time horizons is to exploit hierarchical structure of the problem (Sutton et al., 1998, 1999; Kulkarni et al., 2016; Dayan and Hinton, 1993; Vezhnevets et al., 2017; Dietterich, 2000). Our approach leverages hierarchical structure in imitation learning. We study the case where the underlying problem is hierarchical, and subtasks can be easily elicited from an expert. Our key design principle is an algorithmic framework called *hierarchical guidance*, in which feedback (labels) from the high-level expert is used to focus (guide) the low-level

learner. The high-level expert ensures that low-level learning only occurs when necessary (when subtasks have not been mastered) and only over relevant parts of the state space. This differs from a naïve hierarchical approach which merely gives a subtask decomposition. Focusing on relevant parts of the state space speeds up learning (improves sample efficiency), while omitting feedback on the already mastered subtasks reduces expert effort (improves label efficiency).

We begin by formalizing the problem of hierarchical imitation learning (section 6.3) and carefully separate out cost structures that naturally arise when the expert provides feedback at multiple levels of abstraction. We first apply hierarchical guidance to IL, derive hierarchically guided variants of behavior cloning and DAgger (Ross et al., 2011b), and theoretically analyze the benefits (section 6.4). We next apply hierarchical guidance to the hybrid setting with high-level IL and low-level RL (section 6.5). This architecture is particularly suitable in settings where we have access to high-level semantic knowledge, the subtask horizon is sufficiently short, but the low-level expert is too costly or unavailable. We demonstrate the efficacy of our approaches on a simple but extremely challenging maze domain, and on Montezuma's Revenge (section 6.6). Our experiments show that incorporating a modest amount of expert feedback can lead to dramatic improvements in performance compared to pure hierarchical RL.[1]

6.2 Related Work in Imitation and Reinforcement Learning

For brevity, we provide here a short overview of related work, and defer to Appendix E.3 for additional discussion.

Imitation Learning. One can broadly dichotomize IL into passive collection of demonstrations (behavioral cloning) versus active collection of demonstrations. The former setting (Abbeel and Ng, 2004; Ziebart et al., 2008; Syed and Schapire, 2008; Ho and Ermon, 2016) assumes that demonstrations are collected a priori and the goal of IL is to find a policy that mimics the demonstrations. The latter setting (Daumé III et al., 2009; Ross et al., 2011b; Ross and Bagnell, 2014; Chang et al., 2015; Sun et al., 2017) assumes an interactive expert that provides demonstrations in response to actions taken by the current policy. We explore extension of both approaches into hierarchical settings.

Hierarchical Reinforcement Learning. Several RL approaches to learning hier-

[1] Code and experimental setups are available at `https://sites.google.com/view/hierarchical-il-rl`

archical policies have been explored, foremost among them the options framework (Sutton et al., 1998, 1999; Fruit and Lazaric, 2017). It is often assumed that a useful set of options are fully defined a priori, and (semi-Markov) planning and learning only occurs at the higher level. In comparison, our agent does not have direct access to policies that accomplish such subgoals and has to learn them via expert or reinforcement feedback. The closest hierarchical RL work to ours is that of (Kulkarni et al., 2016), which uses a similar hierarchical structure, but no high-level expert and hence no hierarchical guidance.

Combining Reinforcement and Imitation Learning. The idea of combining IL and RL is not new (Nair et al., 2017; Hester et al., 2018). However, previous work focuses on flat policy classes that use IL as a "pre-training" step (e.g., by pre-populating the replay buffer with demonstrations). In contrast, we consider feedback at multiple levels for a hierarchical policy class, with different levels potentially receiving different types of feedback (i.e., imitation at one level and reinforcement at the other). Somewhat related to our hierarchical expert supervision is the approach of (Andreas et al., 2017), which assumes access to symbolic descriptions of subgoals, without knowing what those symbols mean or how to execute them. Previous literature has not focused much on comparisons of sample complexity between IL and RL, with the exception of the recent work of (Sun et al., 2017).

6.3 Hierarchical Formalism

For simplicity, we consider environments with a natural two-level hierarchy; the HI level corresponds to choosing subtasks, and the LO level corresponds to executing those subtasks. For instance, an agent's overall goal may be to leave a building. At the HI level, the agent may first choose the subtask *"go to the elevator,"* then *"take the elevator down,"* and finally *"walk out."* Each of these subtasks needs to be executed at the LO level by actually navigating the environment, pressing buttons on the elevator, etc.[2]

Subtasks, which we also call *subgoals*, are denoted as $g \in \mathcal{G}$, and the primitive actions are denoted as $a \in \mathcal{A}$. An agent (also referred to as learner) acts by iteratively choosing a subgoal g, carrying it out by executing a sequence of actions a

[2] An important real-world application is in goal-oriented dialogue systems. For instance, a chatbot assisting a user with reservation and booking for flights and hotels (Peng et al., 2017; El Asri et al., 2017) needs to navigate through multiple turns of conversation. The chatbot developer designs the hierarchy of subtasks, such as *ask_user_ goal*, *ask_dates*, *offer_flights*, *confirm*, etc. Each subtask consists of several turns of conversation. Typically a global state tracker exists alongside the hierarchical dialogue policy to ensure that cross-subtask constraints are satisfied.

until completion, and then picking a new subgoal. The agent's choices can depend on an observed state $s \in \mathcal{S}$.[3] We assume that the horizon at the HI level is H_{HI}, i.e., a trajectory uses at most H_{HI} subgoals, and the horizon at the LO level is H_{LO}, i.e., after at most H_{LO} primitive actions, the agent either accomplishes the subgoal or needs to decide on a new subgoal. The total number of primitive actions in a trajectory is thus at most $H_{\text{FULL}} \coloneqq H_{\text{HI}} H_{\text{LO}}$.

The hierarchical learning problem is to simultaneously learn a HI-level policy $\mu : \mathcal{S} \to \mathcal{G}$, called the *meta-controller*, as well as the subgoal policies $\pi_g : \mathcal{S} \to \mathcal{A}$ for each $g \in \mathcal{G}$, called *subpolicies*. The aim of the learner is to achieve a high reward when its meta-controller and subpolicies are run together. For each subgoal g, we also have a (possibly learned) termination function $\beta_g : \mathcal{S} \to \{\mathit{True}, \mathit{False}\}$, which terminates the execution of π_g. The hierarchical agent behaves as follows:

for $h_{\text{HI}} = 1 \ldots H_{\text{HI}}$ **do**
  observe state s and choose subgoal $g \leftarrow \mu(s)$
  for $h_{\text{LO}} = 1 \ldots H_{\text{LO}}$ **do**
    observe state s
    if $\beta_g(s)$ **then break**
    choose action $a \leftarrow \pi_g(s)$
  end for
end for

The execution of each subpolicy π_g generates a *LO-level trajectory* $\tau = (s_1, a_1, \ldots, s_H, a_H, s_{H+1})$ with $H \leqslant H_{\text{LO}}$.[4] The overall behavior results in a *hierarchical trajectory* $\sigma = (s_1, g_1, \tau_1, s_2, g_2, \tau_2, \ldots)$, where the last state of each LO-level trajectory τ_h coincides with the next state s_{h+1} in σ and the first state of the next LO-level trajectory τ_{h+1}. The subsequence of σ which excludes the LO-level trajectories τ_h is called the *HI-level trajectory*, $\tau_{\text{HI}} \coloneqq (s_1, g_1, s_2, g_2, \ldots)$. Finally, the *full trajectory*, τ_{FULL}, is the concatenation of all the LO-level trajectories.

We assume access to an *expert*, endowed with a meta-controller $\mu^\star$, subpolicies $\pi_g^\star$, and termination functions $\beta_g^\star$, who can provide one or several types of supervision:

- `HierDemo`(s): *hierarchical demonstration*. The expert executes its hierarchical policy starting from s and returns the resulting hierarchical trajectory

[3] While we use the term state for simplicity, we do not require the environment to be fully observable or Markovian.

[4] The trajectory might optionally include a reward signal after each primitive action, which might either come from the environment, or be a pseudo-reward as we will see in Section 6.5.

$\sigma^\star = (s_1^\star, g_1^\star, \tau_1^\star, s_2^\star, g_2^\star, \tau_2^\star, \dots)$, where $s_1^\star = s$.

- $\texttt{Label}_{\text{HI}}(\tau_{\text{HI}})$: *HI-level labeling*. The expert provides a good next subgoal at each state of a given HI-level trajectory $\tau_{\text{HI}} = (s_1, g_1, s_2, g_2, \dots)$, yielding a labeled data set $\{(s_1, g_1^\star), (s_2, g_2^\star), \dots\}$.
- $\texttt{Label}_{\text{LO}}(\tau; g)$: *LO-level labeling*. The expert provides a good next primitive action towards a given subgoal g at each state of a given LO-level trajectory $\tau = (s_1, a_1, s_2, a_2, \dots)$, yielding a labeled data set $\{(s_1, a_1^\star), (s_2, a_2^\star), \dots\}$.
- $\texttt{Inspect}_{\text{LO}}(\tau; g)$: *LO-level inspection*. Instead of annotating every state of a trajectory with a good action, the expert only verifies whether a subgoal g was accomplished, returning either *Pass* or *Fail*.
- $\texttt{Label}_{\text{FULL}}(\tau_{\text{FULL}})$: *full labeling*. The expert labels the agent's full trajectory $\tau_{\text{FULL}} = (s_1, a_1, s_2, a_2, \dots)$, from start to finish, ignoring hierarchical structure, yielding a labeled data set $\{(s_1, a_1^\star), (s_2, a_2^\star), \dots\}$.
- $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}})$: *full inspection*. The expert verifies whether the agent's overall goal was accomplished, returning either *Pass* or *Fail*.

When the agent learns not only the subpolicies π_g, but also termination functions β_g, then $\texttt{Label}_{\text{LO}}$ also returns good termination values $\omega^\star \in \{\mathit{True}, \mathit{False}\}$ for each state of $\tau = (s_1, a_1 \dots)$, yielding a data set $\{(s_1, a_1^\star, \omega_1^\star), \dots\}$.

Although `HierDemo` and `Label` can be both generated by the expert's hierarchical policy $(\mu^\star, \{\pi_g^\star\})$, they differ in the mode of expert interaction. `HierDemo` returns a hierarchical trajectory *executed by the expert*, as required for passive IL, and enables a hierarchical version of behavioral cloning (Abbeel and Ng, 2004; Syed and Schapire, 2008). `Label` operations provide labels *with respect to the learning agent's trajectories*, as required for interactive IL. $\texttt{Label}_{\text{FULL}}$ is the standard query used in prior work on learning flat policies (Daumé III et al., 2009; Ross et al., 2011b), and $\texttt{Label}_{\text{HI}}$ and $\texttt{Label}_{\text{LO}}$ are its hierarchical extensions.

`Inspect` operations are newly introduced in this work, and form a cornerstone of our interactive hierarchical guidance protocol that enables substantial savings in label efficiency. They can be viewed as "lazy" versions of the corresponding `Label` operations, requiring less effort. Our underlying assumption is that if the given hierarchical trajectory $\sigma = \{(s_h, g_h, \tau_h)\}$ agrees with the expert on HI level, i.e., $g_h = \mu^\star(s_h)$, and LO-level trajectories pass the inspection, i.e., $\texttt{Inspect}_{\text{LO}}(\tau_h; g_h) = \mathit{Pass}$, then the resulting full trajectory must also pass the full inspection, $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}}) = \mathit{Pass}$.

Algorithm 8 Hierarchical Behavioral Cloning (**h-BC**)

Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \varnothing$ and $\mathcal{D}_g \leftarrow \varnothing$, $g \in \mathcal{G}$
for $t = 1, \ldots, T$ **do**
  Get a new environment instance with start state s
  $\sigma^\star \leftarrow \texttt{HierDemo}(s)$
  for all $(s_h^\star, g_h^\star, \tau_h^\star) \in \sigma^\star$ **do**
    Append $\mathcal{D}_{g_h^\star} \leftarrow \mathcal{D}_{g_h^\star} \cup \tau_h^\star$
    Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \{(s_h^\star, g_h^\star)\}$
  end for
end for
Train subpolicies $\pi_g \leftarrow \texttt{Train}(\pi_g, \mathcal{D}_g)$ for all g
Train meta-controller $\mu \leftarrow \texttt{Train}(\mu, \mathcal{D}_{\text{HI}})$

This means that a hierarchical policy need not always agree with the expert's execution at LO level to succeed in the overall task.

Besides algorithmic reasons, the motivation for separating the types of feedback is that different expert queries will typically require different amount of effort, which we refer to as *cost*. We assume the costs of the `Label` operations are C^L_{HI}, C^L_{LO} and C^L_{FULL}, the costs of each `Inspect` operation are C^I_{LO} and C^I_{FULL}. In many settings, LO-level inspection will require significantly less effort than LO-level labeling, i.e., $C^I_{\text{LO}} \ll C^L_{\text{LO}}$. For instance, identifying if a robot has successfully navigated to the elevator is presumably much easier than labeling an entire path to the elevator. One reasonable cost model, natural for the environments in our experiments, is to assume that `Inspect` operations take time $O(1)$ and work by checking the final state of the trajectory, whereas `Label` operations take time proportional to the trajectory length, which is $O(H_{\text{HI}})$, $O(H_{\text{LO}})$ and $O(H_{\text{HI}}H_{\text{LO}})$ for our three `Label` operations.

6.4 Hierarchically Guided Imitation Learning

Hierarchical guidance is an algorithmic design principle in which the feedback from high-level expert guides the low-level learner in two different ways: (i) the high-level expert ensures that low-level expert is only queried when necessary (when the subtasks have not been mastered yet), and (ii) low-level learning is limited to the relevant parts of the state space. We instantiate this framework first within passive learning from demonstrations, obtaining *hierarchical behavioral cloning* (Algorithm 8), and then within interactive imitation learning, obtaining *hierarchically guided DAgger* (Algorithm 9), our best-performing algorithm.

Algorithm 9 Hierarchically Guided DAgger (**hg-DAgger**)

1: Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \varnothing$ and $\mathcal{D}_g \leftarrow \varnothing$, $g \in \mathcal{G}$
2: Run Hierarchical Behavioral Cloning (Algorithm 8) up to $t = T_{\text{warm-start}}$
3: **for** $t = T_{\text{warm-start}} + 1, \ldots, T$ **do**
4: Get a new environment instance with start state s
5: Initialize $\sigma \leftarrow \varnothing$
6: **repeat**
7: $g \leftarrow \mu(s)$
8: Execute π_g, obtain LO-level trajectory τ
9: Append (s, g, τ) to σ
10: $s \leftarrow$ the last state in τ
11: **until** end of episode
12: Extract τ_{FULL} and τ_{HI} from σ
13: **if** $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}}) = \textit{Fail}$ **then**
14: $\mathcal{D}^\star \leftarrow \texttt{Label}_{\text{HI}}(\tau_{\text{HI}})$
15: Process $(s_h, g_h, \tau_h) \in \sigma$ in sequence as long as g_h agrees with the expert's choice $g_h^\star$ in $\mathcal{D}^\star$:
16: **if** $\texttt{Inspect}(\tau_h; g_h) = \textit{Fail}$ **then**
17: Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \texttt{Label}_{\text{LO}}(\tau_h; g_h)$
18: **break**
19: **end if**
20: Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \mathcal{D}^\star$
21: **end if**
22: Update subpolicies $\pi_g \leftarrow \texttt{Train}(\pi_g, \mathcal{D}_g)$ for all g
23: Update meta-controller $\mu \leftarrow \texttt{Train}(\mu, \mathcal{D}_{\text{HI}})$
24: **end for**

Hierarchical Behavioral Cloning (h-BC)

We consider a natural extension of behavioral cloning to the hierarchical setting (Algorithm 8). The expert provides a set of hierarchical demonstrations $\sigma^\star$, each consisting of LO-level trajectories $\tau_h^\star = \{(s_\ell^\star, a_\ell^\star)\}_{\ell=1}^{H_{\text{LO}}}$ as well as a HI-level trajectory $\tau_{\text{HI}}^\star = \{(s_h^\star, g_h^\star)\}_{h=1}^{H_{\text{HI}}}$. We then run `Train` (lines 8–9) to find the subpolicies π_g that best predict $a_\ell^\star$ from $s_\ell^\star$, and meta-controller μ that best predicts $g_h^\star$ from $s_h^\star$, respectively. `Train` can generally be any supervised learning subroutine, such as stochastic optimization for neural networks or some batch training procedure. When termination functions β_g need to be learned as part of the hierarchical policy, the labels $\omega_g^\star$ will be provided by the expert as part of $\tau_h^\star = \{(s_\ell^\star, a_\ell^\star, \omega_\ell^\star)\}$.[5] In this setting, hierarchical guidance is automatic, because subpolicy demonstrations only occur in relevant parts of the state space.

[5] In our hierarchical imitation learning experiments, the termination functions are all learned. Formally, the termination signal ω_g, can be viewed as part of an augmented action at LO level.

Hierarchically Guided DAgger (hg-DAgger)

Passive IL, e.g., behavioral cloning, suffers from the distribution mismatch between the learning and execution distributions. This mismatch is addressed by interactive IL algorithms, such as SEARN (Daumé III et al., 2009) and DAgger (Ross et al., 2011b), where the expert provides correct actions along the learner's trajectories through the operation $\texttt{Label}_{\text{FULL}}$. A naïve hierarchical implementation would provide correct labels along the entire hierarchical trajectory via $\texttt{Label}_{\text{HI}}$ and $\texttt{Label}_{\text{LO}}$. We next show how to use hierarchical guidance to decrease LO-level expert costs.

We leverage two HI-level query types: $\texttt{Inspect}_{\text{LO}}$ and $\texttt{Label}_{\text{HI}}$. We use $\texttt{Inspect}_{\text{LO}}$ to verify whether the subtasks are successfully completed and $\texttt{Label}_{\text{HI}}$ to check whether we are staying in the relevant part of the state space. The details are presented in Algorithm 9, which uses DAgger as the learner on both levels, but the scheme can be adapted to other interactive imitation learners.

In each episode, the learner executes the hierarchical policy, including choosing a subgoal (line 7), executing the LO-level trajectories, i.e., rolling out the subpolicy π_g for the chosen subgoal, and terminating the execution according to β_g (line 8). Expert only provides feedback when the agent fails to execute the entire task, as verified by $\texttt{Inspect}_{\text{FULL}}$ (line 13). When $\texttt{Inspect}_{\text{FULL}}$ fails, the expert first labels the correct subgoals via $\texttt{Label}_{\text{HI}}$ (line 14), and only performs LO-level labeling as long as the learner's meta-controller chooses the correct subgoal g_h (line 15), but its subpolicy fails (i.e., when $\texttt{Inspect}_{\text{LO}}$ on line 16 fails). Since all the preceding subgoals were chosen and executed correctly, and the current subgoal is also correct, LO-level learning is in the "relevant" part of the state space. However, since the subpolicy execution failed, its learning has not been mastered yet. We next analyze the savings in expert cost that result from hierarchical guidance.

Theoretical Analysis. We analyze the cost of hg-DAgger in comparison with flat DAgger under somewhat stylized assumptions. We assume that the learner aims to learn the meta-controller μ from some policy class $\mathcal{M}$, and subpolicies π_g from some class Π_{LO}. The classes $\mathcal{M}$ and Π_{LO} are finite (but possibly exponentially large) and the task is realizable, i.e., the expert's policies can be found in the corresponding classes: $\mu^\star \in \mathcal{M}$, and $\pi_g^\star \in \Pi_{\text{LO}}$, $g \in \mathcal{G}$. This allows us to use the *halving algorithm* (Shalev-Shwartz et al., 2012) as the online learner on both levels. (The implementation of our algorithm does not require these assumptions.)

The halving algorithm maintains a version space over policies, acts by a majority decision, and when it makes a mistake, it removes all the erring policies from the

version space. In the hierarchical setting, it therefore makes at most $\log|\mathcal{M}|$ mistakes on the HI level, and at most $\log|\Pi_{\text{LO}}|$ mistakes when learning each π_g. The mistake bounds can be further used to upper bound the total expert cost in both hg-DAgger and flat DAgger. To enable an apples-to-apples comparison, we assume that the flat DAgger learns over the policy class $\Pi_{\text{FULL}} = \{(\mu, \{\pi_g\}_{g\in\mathcal{G}}) : \mu \in \mathcal{M}, \pi_g \in \Pi_{\text{LO}}\}$, but is otherwise oblivious to the hierarchical task structure. The bounds depend on the cost of performing different types of operations, as defined at the end of Section 6.3. We consider a modified version of flat DAgger that first calls $\texttt{Inspect}_{\text{FULL}}$, and only requests labels ($\texttt{Label}_{\text{FULL}}$) if the inspection fails. The proofs are deferred to Appendix E.1.

Theorem 6.4.1. *Given finite classes $\mathcal{M}$ and Π_{LO} and realizable expert policies, the total cost incurred by the expert in hg-DAgger by round T is bounded by*

$$TC^I_{FULL} + \left(\log_2|\mathcal{M}| + |\mathcal{G}_{opt}|\log_2|\Pi_{LO}|\right)(C^L_{HI} + H_{HI}C^I_{LO}) + \left(|\mathcal{G}_{opt}|\log_2|\Pi_{LO}|\right)C^L_{LO}, \tag{6.1}$$

where $\mathcal{G}_{opt} \subseteq \mathcal{G}$ is the set of the subgoals actually used by the expert, $\mathcal{G}_{opt} \coloneqq \mu^\star(\mathcal{S})$.

Theorem 6.4.2. *Given the full policy class $\Pi_{FULL} = \{(\mu, \{\pi_g\}_{g\in\mathcal{G}}) : \mu \in \mathcal{M}, \pi_g \in \Pi_{LO}\}$ and a realizable expert policy, the total cost incurred by the expert in flat DAgger by round T is bounded by*

$$TC^I_{FULL} + \left(\log_2|\mathcal{M}| + |\mathcal{G}|\log_2|\Pi_{LO}|\right)C^L_{FULL}. \tag{6.2}$$

Both bounds have the same leading term, TC^I_{FULL}, the cost of full inspection, which is incurred every round and can be viewed as the "cost of monitoring." In contrast, the remaining terms can be viewed as the "cost of learning" in the two settings, and include terms coming from their respective mistake bounds. The ratio of the cost of hierarchically guided learning to the flat learning is then bounded as

$$\frac{\text{Eq. (6.1)} - TC^I_{\text{FULL}}}{\text{Eq. (6.2)} - TC^I_{\text{FULL}}} \leqslant \frac{C^L_{\text{HI}} + H_{\text{HI}}C^I_{\text{LO}} + C^L_{\text{LO}}}{C^L_{\text{FULL}}}, \tag{6.3}$$

where we applied the upper bound $|\mathcal{G}_{\text{opt}}| \leqslant |\mathcal{G}|$. The savings thanks to hierarchical guidance depend on the specific costs. Typically, we expect the inspection costs to be $O(1)$, if it suffices to check the final state, whereas labeling costs scale linearly with the length of the trajectory. The cost ratio is then $\propto \frac{H_{\text{HI}}+H_{\text{LO}}}{H_{\text{HI}}H_{\text{LO}}}$. Thus, we realize most significant savings if the horizons on each individual level are substantially shorter than the overall horizon. In particular, if $H_{\text{HI}} = H_{\text{LO}} = \sqrt{H_{\text{FULL}}}$, the hierarchically

Algorithm 10 Hierarchically Guided DAgger / Q-learning

input Function $\texttt{pseudo}(s; g)$ providing the pseudo-reward
input Predicate $\texttt{terminal}(s; g)$ indicating the termination of g
input Annealed exploration probabilities $\epsilon_g > 0$, $g \in \mathcal{G}$
Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \varnothing$ and $\mathcal{D}_g \leftarrow \varnothing$, $g \in \mathcal{G}$
Initialize subgoal Q-functions Q_g, $g \in \mathcal{G}$
for $t = 1, \ldots, T$ **do**
  Get a new environment instance with start state s
  Initialize $\sigma \leftarrow \varnothing$
  repeat
    $s_{\text{HI}} \leftarrow s$, $g \leftarrow \mu(s)$ and initialize $\tau \leftarrow \varnothing$
    repeat
      $a \leftarrow \epsilon_g$-greedy(Q_g, s)
      Execute a, next state $\tilde{s}$, $\tilde{r} \leftarrow \texttt{pseudo}(\tilde{s}; g)$
      Update Q_g: a (stochastic) gradient descent step on a minibatch from $\mathcal{D}_g$
      Append $(s, a, \tilde{r}, \tilde{s})$ to τ and update $s \leftarrow \tilde{s}$
    until $\texttt{terminal}(s; g)$
    Append (s_{HI}, g, τ) to σ
  until end of episode
  Extract τ_{FULL} and τ_{HI} from σ
  if $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}}) = \textit{Fail}$ **then**
    $\mathcal{D}^\star \leftarrow \texttt{Label}_{\text{HI}}(\tau_{\text{HI}})$
    Process $(s_h, g_h, \tau_h) \in \sigma$ in sequence as long as g_h agrees with the expert's choice $g_h^\star$ in $\mathcal{D}^\star$:
      Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \tau_h$
    Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \mathcal{D}^\star$
  else
    Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \tau_h$ for all $(s_h, g_h, \tau_h) \in \sigma$
  end if
  Update meta-controller $\mu \leftarrow \texttt{Train}(\mu, \mathcal{D}_{\text{HI}})$
end for

guided approach reduces the overall labeling cost by a factor of $\sqrt{H_{\text{FULL}}}$. More generally, whenever H_{FULL} is large, we reduce the costs of learning be at least a constant factor—a significant gain if this is a saving in the effort of a domain expert.

6.5 Hierarchically Guided IL / RL

Hierarchical guidance also applies in the hybrid setting with interactive IL on the HI level and RL on the LO level. The HI-level expert provides the hierarchical decomposition, including the pseudo-reward function for each subgoal,[6] and is also

[6] This is consistent with many hierarchical RL approaches, including options (Sutton et al., 1999), MAXQ (Dietterich, 2000), UVFA (Schaul et al., 2015a) and h-DQN (Kulkarni et al., 2016).

able to pick a correct subgoal at each step. Similar to hg-DAgger, the labels from HI-level expert are used not only to train the meta-controller μ, but also to limit the LO-level learning to the relevant part of the state space. In Algorithm 10 we provide the details, with DAgger on HI level and Q-learning on LO level. The scheme can be adapted to other interactive IL and RL algorithms.

The learning agent proceeds by *rolling in* with its meta-controller (line 7). For each selected subgoal g, the subpolicy π_g selects and executes primitive actions via the ϵ-greedy rule (lines 9–10), until some termination condition is met. The agent receives some pseudo-reward, also known as intrinsic reward (Kulkarni et al., 2016) (line 10). Upon termination of the subgoal, agent's meta-controller μ chooses another subgoal and the process continues until the end of the episode, where the involvement of the expert begins. As in hg-DAgger, the expert inspects the overall execution of the learner (line 17), and if it is not successful, the expert provides HI-level labels, which are accumulated for training the meta-controller.

Hierarchical guidance impacts how the LO-level learners accumulate experience. As long as the meta-controller's subgoal g agrees with the expert's, the agent's experience of executing subgoal g is added to the experience replay buffer $\mathcal{D}_g$. If the meta-controller selects a "bad" subgoal, the accumulation of experience in the current episode is terminated. This ensures that experience buffers contain only the data from the relevant part of the state space.

Algorithm 10 assumes access to a real-valued function $\texttt{pseudo}(s; g)$, providing the pseudo-reward in state s when executing g, and a predicate $\texttt{terminal}(s; g)$, indicating the termination (not necessarily successful) of subgoal g. This setup is similar to prior work on hierarchical RL (Kulkarni et al., 2016). One natural definition of pseudo-rewards, based on an additional predicate $\texttt{success}(s; g)$ indicating a successful completion of subgoal g, is as follows:

$$\begin{cases} 1 & \text{if } \texttt{success}(s; g) \\ -1 & \text{if } \neg\texttt{success}(s; g) \text{ and } \texttt{terminal}(s; g) \\ -\kappa & \text{otherwise,} \end{cases}$$

where $\kappa > 0$ is a small penalty to encourage short trajectories. The predicates `success` and `terminal` are provided by an expert or learnt from supervised or reinforcement feedback. In our experiments, we explicitly provide these predicates to both hg-DAgger/Q as well as the hierarchical RL, giving them advantage over hg-DAgger, which needs to learn when to terminate subpolicies.

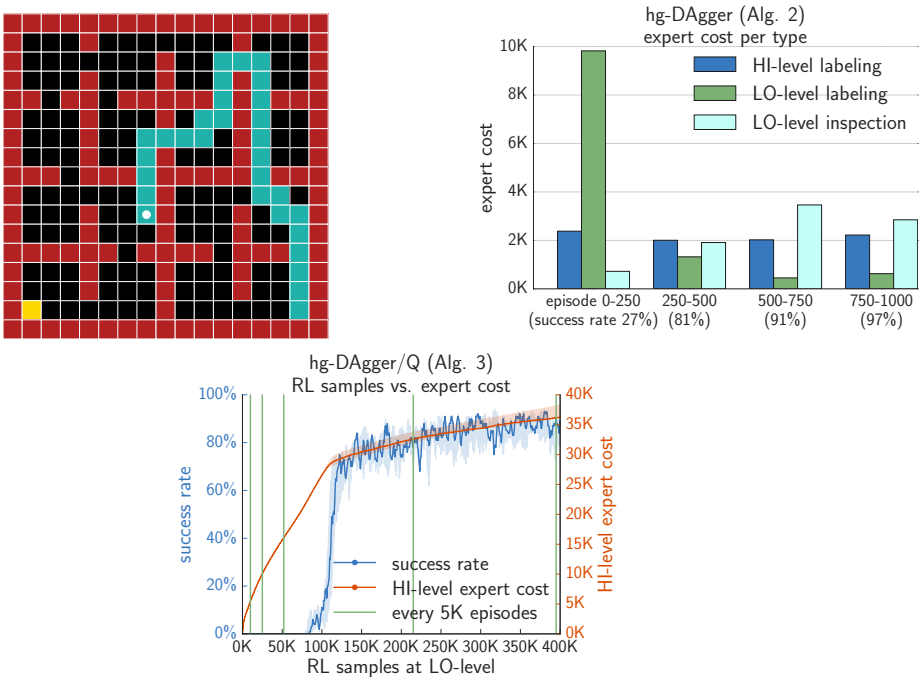


Figure 6.1: *Maze navigation*. *(Left)* One sampled environment instance; the agent needs to navigate from bottom right to bottom left. *(Middle)* Expert cost over time for hg-DAgger; the cost of `Label` operations equals the length of labeled trajectory, the cost of `Inspect` operations is 1. *(Right)* Success rate of hg-DAgger/Q and the HI-level label cost as a function of the number of LO-level RL samples.

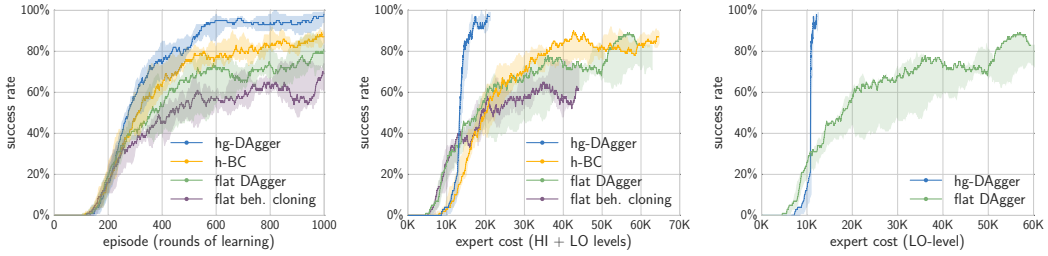


Figure 6.2: *Maze navigation: hierarchical versus flat imitation learning*. Each episode is followed by a round of training and a round of testing. The success rate is measured over previous 100 test episodes; the expert cost is as in Figure 6.1. *(Left)* Success rate per episode. *(Middle)* Success rate versus the expert cost. *(Right)* Success rate versus the LO-level expert cost.

6.6 Experiments

We evaluate the performance of our algorithms on two separate domains: (i) a simple but challenging maze navigation domain and (ii) the Atari game Montezuma's Revenge.

Maze Navigation Domain

Task Overview. Figure 6.1 (left) displays a snapshot of the maze navigation domain. In each episode, the agent encounters a new instance of the maze from a large collection of different layouts. Each maze consists of 16 rooms arranged in a 4-by-4 grid, but the openings between the rooms vary from instance to instance as does the initial position of the agent and the target. The agent (white dot) needs to navigate from one corner of the maze to the target marked in yellow. Red cells are obstacles (lava), which the agent needs to avoid for survival. The contextual information the agent receives is the pixel representation of a bird's-eye view of the environment, including the partial trail (marked in green) indicating the visited locations.

Due to a large number of random environment instances, this domain is not solvable with tabular algorithms. Note that rooms are not always connected, and the locations of the hallways are not always in the middle of the wall. Primitive actions include going one step *up*, *down*, *left* or *right*. In addition, each instance of the environment is designed to ensure that there is a path from initial location to target, and the shortest path takes at least 45 steps ($H_{\text{FULL}} = 100$). The agent is penalized with reward -1 if it runs into lava, which also terminates the episode. The agent only receives positive reward upon stepping on the yellow block.

A hierarchical decomposition of the environment corresponds to four possible subgoals of going to the room immediately to the *north*, *south*, *west*, *east*, and the fifth possible subgoal *go_to_target* (valid only in the room containing the target). In this setup, $H_{\text{LO}} \approx 5$ steps, and $H_{\text{HI}} \approx$ 10–12 steps. The episode terminates after 100 primitive steps if the agent is unsuccessful. The subpolicies and meta-controller use similar neural network architectures and only differ in the number of action outputs. (Details of network architecture are provided in Appendix E.2.)

Hierarchically Guided IL. We first compare our hierarchical IL algorithms with their flat versions. The algorithm performance is measured by success rate, defined as the average rate of successful task completion over the previous 100 test episodes, on random environment instances not used for training. The cost of each `Label` operation equals the length of the labeled trajectory, and the cost of each `Inspect` operation equals 1.

Both h-BC and hg-DAgger outperform flat imitation learners (6.2,left). hg-DAgger, in particular, achieves consistently the highest success rate, approaching 100% in fewer than 1000 episodes. 6.2 (left) displays the median as well as the range from minimum to maximum success rate over 5 random executions of the algorithms.

Expert cost varies significantly between the two hierarchical algorithms. Figure 6.2 (middle) displays the same success rate, but as a function of the expert cost. hg-DAgger achieves significant savings in expert cost compared to other imitation learning algorithms thanks to a more efficient use of the LO-level expert through hierarchical guidance. Figure 6.1 (middle) shows that hg-DAgger requires most of its LO-level labels early in the training and requests primarily HI-level labels after the subgoals have been mastered. As a result, hg-DAgger requires only a fraction of LO-level labels compared to flat DAgger (Figure 6.2, right).

Hierarchically Guided IL / RL. We evaluate hg-DAgger/Q with deep double Q-learning (DDQN, Van Hasselt et al., 2016) and prioritized experience replay (Schaul et al., 2015b) as the underlying RL procedure. Each subpolicy learner receives a pseudo-reward of 1 for each successful execution, corresponding to stepping through the correct door (e.g., door to the north if the subgoal is *north*) and negative reward for stepping into lava or through other doors.

6.1 (right) shows the learning progression of hg-DAgger/Q, implying two main observations. First, the number of HI-level labels rapidly increases initially and then flattens out after the learner becomes more successful, thanks to the availability of $\texttt{Inspect}_{\text{FULL}}$ operation. As the hybrid algorithm makes progress and the learning agent passes the $\texttt{Inspect}_{\text{FULL}}$ operation increasingly often, the algorithm starts saving significantly on expert feedback. Second, the number of HI-level labels is higher than for both hg-DAgger and h-BC. $\texttt{Inspect}_{\text{FULL}}$ returns *Fail* often, especially during the early parts of training. This is primarily due to the slower learning speed of Q-learning at the LO level, requiring more expert feedback at the HI level. This means that the hybrid algorithm is suited for settings where LO-level expert labels are either not available or more expensive than the HI-level labels. This is exactly the setting we analyze in the next section.

In Appendix E.2, we compare hg-DAgger/Q with hierarchical RL (h-DQN, Kulkarni et al., 2016), concluding that h-DQN, even with significantly more LO-level samples, fails to reach success rate comparable to hg-DAgger/Q. Flat Q-learning also fails in this setting, due to a long planning horizon and sparse rewards (Mnih et al., 2015).

Hierarchically Guided IL / RL vs Hierarchical RL: Comparison on Montezuma's Revenge

Task Overview. Montezuma's Revenge is among the most difficult Atari games for existing deep RL algorithms, and is a natural candidate for hierarchical approach

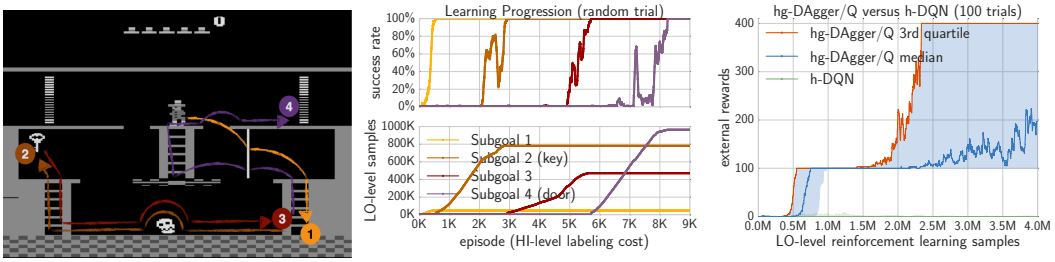


Figure 6.3: *Montezuma's revenge: hg-DAgger/Q versus h-DQN. (Left)* Screenshot of Montezuma's Revenge in black-and-white with color-coded subgoals. *(Middle)* Learning progression of hg-DAgger/Q in solving the first room of Montezuma's Revenge for a typical successful trial. Subgoal colors match the left pane; success rate is the fraction of times the LO-level RL learner achieves its subgoal over the previous 100 attempts. *(Right)* Learning performance of hg-DAgger/Q versus h-DQN (median and inter-quartile range).

due to the sequential order of subtasks. Figure 6.3 (left) displays the environment and an annotated sequence of subgoals. The four designated subgoals are: go to bottom of the right stair, get the key, reverse path to go back to the right stair, then go to open the door (while avoiding obstacles throughout).

The agent is given a pseudo-reward of 1 for each subgoal completion and -1 upon loss of life. We enforce that the agent can only have a single life per episode, preventing the agent from taking a shortcut after collecting the key (by taking its own life and re-initializing with a new life at the starting position, effectively collapsing the task horizon). Note that for this setting, the actual game environment is equipped with two positive external rewards corresponding to picking up the key (subgoal 2, reward of 100) and using the key to open the door (subgoal 4, reward of 300). Optimal execution of this sequence of subgoals requires more than 200 primitive actions. Unsurprisingly, flat RL algorithms often achieve a score of 0 on this domain (Mnih et al., 2015, 2016; Wang et al., 2016).

hg-DAgger/Q versus h-DQN. Similar to the maze domain, we use DDQN with prioritized experience replay at the LO level of hg-DAgger/Q. We compare its performance with h-DQN using the same neural network architecture as (Kulkarni et al., 2016). Figure 6.3 (middle) shows the learning progression of our hybrid algorithm. The HI-level horizon $H_{\text{HI}} = 4$, so meta-controller is learnt from fairly few samples. Each episode roughly corresponds to one $\texttt{Label}_{\text{HI}}$ query. Subpolicies are learnt in the order of subgoal execution as prescribed by the expert.

We introduce a simple modification to Q-learning on the LO level to speed up learning: the accumulation of experience replay buffer does not begin until the first

time the agent encounters positive pseudo-reward. During this period, in effect, only the meta-controller is being trained. This modification ensures the reinforcement learner encounters at least some positive pseudo-rewards, which boosts learning in the long horizon settings and should naturally work with any off-policy learning scheme (DQN, DDQN, Dueling-DQN). For a fair comparison, we introduce the same modification to the h-DQN learner (otherwise, h-DQN failed to achieve any reward).

To mitigate the instability of DQN (see, for example, learning progression of subgoal 2 and 4 in Figure 6.3, middle), we introduce one additional modification. We terminate training of subpolicies when the success rate exceeds 90%, at which point the subgoal is considered learned. Subgoal success rate is defined as the percentage of successful subgoal completions over the previous 100 attempts.

Figure 6.3 (right) shows the median and the inter-quartile range over 100 runs of hg-DAgger/Q and hg-DQN.[7] The LO-level sample sizes are not directly comparable with the middle panel, which displays the learning progression for a random successful run, rather than an aggregate over multiple runs. In all of our experiments, the performance of the imitation learning component is stable across many different trials, whereas the performance of the reinforcement learning component varies substantially. Subgoal 4 (door) is the most difficult to learn due to its long horizon whereas subgoals 1–3 are mastered very quickly, especially compared to h-DQN. Our algorithm benefits from hierarchical guidance and accumulates experience for each subgoal only within the relevant part of the state space, where the subgoal is part of an optimal trajectory. In contrast, h-DQN may pick bad subgoals and the resulting LO-level samples then "corrupt" the subgoal experience replay buffers and substantially slow down convergence.[8]

The number of HI-level labels in Figure 6.3 (middle) can be further reduced by using a more efficient RL procedure than DDQN at the LO level. In the specific example of Montezuma's Revenge, the actual human effort is in fact much smaller, since the human expert needs to provide a sequence of subgoals only once (together with simple subgoal detectors), and then HI-level labeling can be done automatically. The human expert only needs to understand the high level semantics, and does not need to be able to play the game.

[7] In Appendix E.2, we present additional plots, including 10 best runs of each algorithm, subgoal completion rate over 100 trials, and versions of Figure 6.3 (middle) for additional random instances.

[8] In fact, we further reduced the number of subgoals of h-DQN to only two initial subgoals, but the agent still largely failed to learn even the second subgoal (see the appendix for details).

6.7 Discussion

We have presented *hierarchical guidance* framework and shown how it can be used to speed up learning and reduce the cost of expert feedback in hierarchical imitation learning and hybrid imitation–reinforcement learning.

Our approach can be extended in several ways. For instance, one can consider weaker feedback such as preference or gradient-style feedback (Fürnkranz et al., 2012; Loftin et al., 2016; Christiano et al., 2017), or a weaker form of imitation feedback, only saying whether the agent action is correct or incorrect, corresponding to bandit variant of imitation learning (Ross et al., 2011b).

Our hybrid IL / RL approach relied on the availability of a subgoal termination predicate indicating when the subgoal is achieved. While in many settings such a termination predicate is relatively easy to specify, in other settings this predicate needs to be learned. We leave the question of learning the termination predicate, while learning to act from reinforcement feedback, open for future research.

Chapter 7

POLICY LEARNING WITH LATENT MODEL

Summary. We study the problem of imitation learning from demonstrations of multiple coordinating agents. One key challenge in this setting is that learning a good model of coordination can be difficult, since coordination is often implicit in the demonstrations and must be inferred as a latent variable. We propose a joint approach that simultaneously learns a latent coordination model along with the individual policies. In particular, our method integrates unsupervised structure learning with conventional imitation learning. We illustrate the power of our approach on a difficult problem of learning multiple policies for fine-grained behavior modeling in team sports, where different players occupy different roles in the coordinated team strategy. We show that having a coordination model to infer the roles of players yields substantially improved imitation loss compared to conventional baselines.

7.1 Motivating Applications for Latent Structure

The areas of multi-agent planning and control have witnessed a recent wave of strong interest due to the practical desire to deal with complex real-world problems, such as smart-grid control, autonomous vehicles planning, managing teams of robots for emergency response, among others. From the learning perspective, (cooperative) multi-agent learning is not a new area of research (Stone and Veloso, 2000; Panait and Luke, 2005). However, compared to the progress in conventional supervised learning and single-agent reinforcement learning, the successes of multi-agent learning have remained relatively modest. Most notably, multi-agent learning suffers from extremely high dimensionality of both the state and actions spaces, as well as relative lack of data sources and experimental testbeds.

The growing availability of data sources for coordinated multi-agent behavior, such as sports tracking data (Bialkowski et al., 2014), now enables the possibility of learning multi-agent policies from demonstrations, also known as multi-agent imitation learning. One particularly interesting aspect of domains such as team sports is that the agents must coordinate. For example, in the professional soccer setting depicted in Figure 7.1, different players must coordinate to assume different roles (e.g., defend left field). However, the roles and role assignment mechanism are unobserved from the demonstrations. Furthermore, the role for a player may change

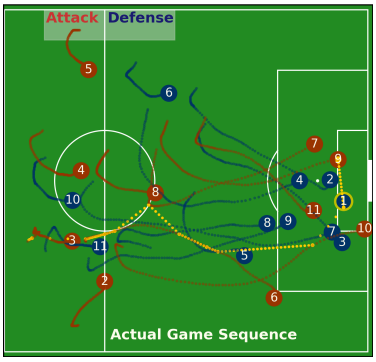


Figure 7.1: *Our motivating example of learning coordinating behavior policies for team sports from tracking data. Red is the attacking team, blue is the defending team, and yellow is the ball.*

during the same play sequence. In the control community, this issue is known as "index-free" multi-agent control (Kingston and Egerstedt, 2010).

Motivated by these challenges, we study the problem of imitation learning for multiple coordinating agents from demonstrations. Many realistic multi-agent settings require coordination among collaborative agents to achieve some common goal (Guestrin et al., 2002; Kok et al., 2003). Beyond team sports, other examples include learning policies for game AI, controlling teams of multiple robots, or modeling collective animal behavior. As discussed above, we are interested in settings where agents have access to the outcome of actions from other agents, but the coordination mechanism is neither clearly defined nor observed, which makes the full state only partially observable.

We propose a semi-supervised learning framework that integrates and builds upon conventional imitation learning and unsupervised, or latent, structure learning. The latent structure model encodes a coordination mechanism, which approximates the implicit coordination in the demonstration data. In order to make learning tractable, we develop an alternating optimization method that enables integrated and efficient training of both individual policies and the latent structure model. For learning individual policies, we extend reduction-based single-agent imitation learning approaches into multi-agent domain, utilizing powerful black-box supervised techniques such as deep learning as base routines. For latent structure learning, we develop a stochastic variational inference approach.

We demonstrate the effectiveness of our method in two settings. The first is a synthetic experiment based on the popular predator-prey game. The second is a challenging task of learning multiple policies for team defense in professional

soccer, using a large training set[1] of play sequences illustrated by Figure 7.1. We show that learning a good latent structure to encode implicit coordination yields significantly superior imitation performance compared to conventional baselines. To the best of our knowledge, this is the first time an imitation learning approach has been applied to jointly learn cooperative multi-agent policies at large scale.

7.2 Policy Learning Problem Formulation

In coordinated multi-agent imitation learning, we have K agents acting in coordination to achieve a common goal (or sequence of goals). Training data $\mathcal{D}$ consists of multiple demonstrations of K agents. Importantly, we assume the identity (or indexing) of the K experts may change from one demonstration to another. Each (unstructured) set of demonstrations is denoted by $U = \{U_1, \ldots, U_K\}$, where $U_k = \{u_{t,k}\}_{t=1}^{T}$ is the sequence of actions by agent k at time t. Note that each set of demonstrations can have varying sequence length T. Let $C = \{c_t\}_{t=1}^{T}$ be the context associated with each demonstration sequence.

Policy Learning. Our ultimate goal is to learn a (largely) decentralized policy, but for clarity we first present the problem of learning a fully centralized multi-agent policy. Following the notation of (Ross et al., 2011a), let $\vec{\pi}(\vec{s}) := \vec{a}$ denote the joint policy that maps the joint state, $\vec{s} = [s_1, \ldots, s_K]$, of all K agents into K actions $\vec{a} = [a_1, \ldots, a_K]$. The goal is to minimize imitation loss:

$$\mathcal{L}_{imitation} = \mathbb{E}_{\vec{s} \sim d_{\vec{\pi}}} \left[\ell(\vec{\pi}(\vec{s})) \right],$$

where $d_{\vec{\pi}}$ denotes the distribution of states experienced by joint policy $\vec{\pi}$ and ℓ is the imitation loss defined over the demonstrations (e.g., squared loss for deterministic policies, or cross entropy for stochastic policies).

The decentralized setting decomposes the joint policy $\vec{\pi} = [\pi_1, \ldots, \pi_K]$ into K policies, each tailored to a specific agent index or "role".[2] The loss function is then:

$$\mathcal{L}_{imitation} = \sum_{k=1}^{K} \mathbb{E}_{s \sim d_{\pi_k}} \left[\ell(\pi_k(s_k)) \right].$$

Black-Box Policy Classes. In order to leverage powerful black-box policy classes such as random forests and deep learning, we take a learning reduction approach to

[1] Data at `http://www.stats.com/data-science/` and see video result at `http://hoangminhle.github.io`

[2] It is straightforward to extend our formulation to settings where multiple agents can occupy the same role, and where not all roles are occupied across all execution sequences.

training $\vec{\pi}$. One consequence is that the state space representation $s = [s_1, \ldots, s_K]$ must be consistently indexed, e.g., agent k in one instance must correspond to agent k in another instance. This requirement applies for both centralized and decentralized policy learning, and is often implicitly assumed in prior work on multi-agent learning. A highly related issue arises in distributed control of index-free coordinating robots, e.g., to maintain a defined formation (Kloder and Hutchinson, 2006; Kingston and Egerstedt, 2010).

Motivating example: Soccer Domain. Consider the task of imitating professional soccer players, where training data includes play sequences from different teams and games. Context C corresponds to the behavior of the opposing team and the ball. The data includes multiple sequences of K-set of trajectories $U = \{U_1, U_2, \ldots, U_K\}$, where the actual identity of player generating U_k may change from one demonstration to the next.

One important challenge for black-box policy learning is constructing an indexing mechanism over the agents to yield a consistent state representation. For example, the same index should correspond to the "left defender" in all instances. Otherwise, the inputs to the policy will be inconsistent, making learning difficult if not impossible. Note that barring extensive annotations or some heuristic rule-based definitions, it is unnatural to quantitatively define what makes a player "left defender". In addition, even if we had a way to define who the "left defender" is, he may not stay in the same role during the same sequence.

Role-based Indexing. We address index-free policy learning via role learning and role-based index assignment. To motivate our notion of role, let's first consider the simplest indexing mechanism: one could equate role to agent identity. However, the data often comes from various sequences, with heterogeneous identities and teams of agents. Thus instead of learning identity-specific policies, it is more natural and data-efficient to learn a policy per role. However, a key challenge in learning policies directly is that *the roles are undefined, unobserved, and could change dynamically within the same sequence.* We thus view learning the coordination, via role assignment, as an unsupervised structured prediction problem.

Coordination via Structured Role Assignment. Instead of handcrafting the definition of roles, we learn the roles in an unsupervised fashion, without attaching any semantic labels to the roles. At the same time, role transition should obey certain structural regularity, due to coordination. This motivates using graphical models to represent the coordination structure.

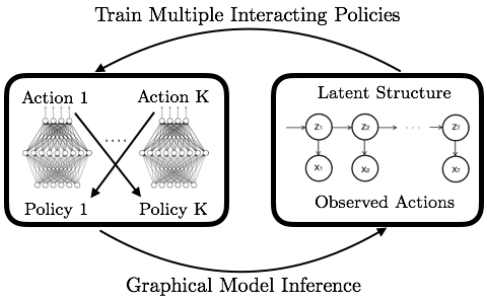


Figure 7.2: *Alternating stochastic optimization training scheme for our semi-supervised structure regularization model.*

Coordinated Policy Learning. We formulate the indexing mechanism as an assignment function $\mathcal{A}$ which maps the unstructured set U and some probabilistic structured model q to an indexed set of trajectory A rearranged from U, i.e.,

$$\mathcal{A} : \{U_1, .., U_K\} \times q \mapsto [A_1, .., A_K],$$

where the set $\{A_1, .., A_K\} \equiv \{U_1, .., U_K\}$. We view q as a latent variable model that infers the role assignments for each set of demonstrations. Thus, q drives the indexing mechanism $\mathcal{A}$ so that state vectors can be consistently constructed to facilitate optimizing for the imitation loss.

We employ entropy regularization, augmenting the imitation loss with some low entropy penalty (Grandvalet et al., 2004; Dudik et al., 2004), yielding our overall objective:

$$\min_{\pi_1,..,\pi_K,\mathcal{A}} \sum_{k=1}^{K} \mathbb{E}_{s_k \sim d_{\pi_k}} \left[\ell(\pi_k(s_k))|\mathcal{A}, \mathcal{D}\right] - \lambda H(\mathcal{A}|\mathcal{D}) \tag{7.1}$$

where both imitation loss and entropy are measured with respect to the state distribution induced by the policies, and $\mathcal{D}$ is training data. This objective can also be seen as maximizing the mutual information between latent structure and observed trajectories (Krause et al., 2010).

7.3 Policy Learning Algorithm with Structure Learning and Inference

Optimizing (7.1) is challenging for two reasons. First, beyond the challenges inherited from single-agent settings, multi-agent imitation learning must account for multiple simultaneously learning agents, which is known to cause non-stationarity for multi-agent reinforcement learning (Busoniu et al., 2008). Second, the latent role assignment model, which forms the basis for coordination, depends on the actions of the learning policies, which in turn depend on the structured role assignment.

Algorithm 11 Coordinated Multi-Agent Imitation Learning

Input: Multiple unstructured trajectory sets $U = \{U_1, \ldots, U_K\}$ with $U_k = \{u_{t,k}\}_{t=1}^{T}$ and context $C = \{c_t\}_{t=1}^{T}$.
Input: Graphical model q with global/local parameters θ and z.
Input: Initialized policies $\pi_k, k = 1, \ldots, K$
Input: Step size sequence $\rho_n, n = 1, 2, \ldots$

1: **repeat**
2: $[A_1, \ldots, A_K] \leftarrow \texttt{Assign}\{U_1, \ldots, U_K | q(\theta, z)\}$
3: $[\pi_1, \ldots, \pi_K] \leftarrow \texttt{Learn}[A_1, \ldots, A_K, C]$
4: Roll-out $\pi_1, \ldots, \pi_K$ to obtain $\widehat{A}_1, \ldots, \widehat{A}_K$
5: $A_k \leftarrow \widehat{A}_k \ \ \forall k$
(Alternatively: $A_k \leftarrow \widehat{A}_k$ with prob η for $\eta \rightarrow 1$)
6: $q(\theta, z) \leftarrow \texttt{LearnStructure}\{A_1, \ldots, A_K, C, \theta, \rho_n\}$
7: **until** No improvement on validation set

output K policies $\pi_1, \pi_2, \ldots, \pi_K$

We propose an alternating optimization approach to solving (7.1), summarized in Figure 7.2. The main idea is to integrate imitation learning with unsupervised structure learning by taking turns to (i) optimize for imitation policies while fixing a structured model (minimizing imitation loss), and (ii) re-train the latent structure model and reassign roles while fixing the learning policies (maximizing role assignment entropy). The alternating nature allows us to circumvent the circular dependency between policy learning and latent structure learning. Furthermore, for (i) we develop a stable multi-agent learning reduction approach.

Approach Outline

Algorithm 11 outlines our framework. We assume the latent structure model for computing role assignments is formulated as a graphical model. The multi-agent policy training procedure `Learn` utilizes a reduction approach, and can leverage powerful off-the-shelf supervised learning tools such as deep neural networks (Hochreiter and Schmidhuber, 1997). The structure learning `LearnStructure` and role assignment `Assign` components are based on graphical model training and inference. For efficient training, we employ alternating stochastic optimization (Hoffman et al., 2013; Johnson and Willsky, 2014; Beal, 2003) on the same mini-batches. Note that batch training can be deployed similarly, as illustrated by one of our experiments.

We interleave the three components described above into a complete learning algorithm. Given an initially unstructured set of training data, an initialized set of policies, and prior parameters of the structure model, Algorithm 11 performs alternating structure optimization on each mini-batch (size 1 in Algorithm 11).

- Line 2: Role assignment is performed on trajectories $\{A_1, \ldots, A_K\}$ by running inference procedure (Algorithm 14). The result is an ordered set $[A_1, \ldots, A_K]$, where trajectory A_k corresponds to policy π_k.
- Line 3-5: Each policy π_k is updated using joint multi-agent training on the ordered set $[A_1, \ldots, A_K, C]$ (Algorithm 12). The updated models are executed to yield a rolled-out set of trajectories, which replace the previous set of trajectories $\{A_k\}$.
- Line 6: Parameters of latent structured model are updated from the rolled-out trajectories (Algorithm 13).

The algorithm optionally includes a mixing step on line 5, where the rolled-out trajectories may replace the training trajectories with increasing probability approaching 1, which is similar to scheduled sampling (Bengio et al., 2015), and may help stabilize learning in the early phase of the algorithm. In our main experiment, we do not notice a performance gain using this option.

Joint Multi-Agent Imitation Learning

In this section we describe the `Learn` procedure for multi-agent imitation learning in Line 3 of Algorithm 11. As background, for single agent imitation learning, reduction-based methods operate by iteratively collecting a new data set $\mathcal{D}_n$ at each round n of training, consisting of state-action pairs (s_t, a_t^*) where a_t^* is some optimal or demonstrated action given state s_t. A new policy can be formed by (i) combining a new policy from this data set $\mathcal{D}_n$ with previously learned policy π (Daumé III et al., 2009) or (ii) learning a new policy π directly from the data set formed by aggregating $\mathcal{D}_1, \ldots, \mathcal{D}_n$ (Ross et al., 2011a). Other variants exist although we do not discuss them here.

The intuition behind the iterative reduction approach is to prevent a mismatch in training and prediction distributions due to sequential cascading errors (also called covariate-shift). The main idea is to use the learned policy's own predictions in the construction of subsequent states, thus simulating the test-time performance during training. This mechanism enables the agent to learn a policy that is robust to its own mistakes. Reduction-based methods also accommodate any black-box supervised training subroutine. We focus on using expressive function classes such as Long

Algorithm 12 Joint Multi-Agent Imitation Learning: $\texttt{Learn}(A_1, A_2, \ldots, A_K, C)$

Input: Ordered actions $A_k = \{a_{t,k}\}_{t=1}^{T} \;\; \forall k$, context $\{c_t\}_{t=1}^{T}$
Input: Initialized policies $\pi_1, \ldots, \pi_K$
Input: base routine $\texttt{Train}(S, A)$ mapping state to actions
1: Set increasing prediction horizon $j \in \{1, \ldots, T\}$
2: **for** $t = 0, j, 2j, \ldots, T$ **do**
3: **for** $i = 0, 1, \ldots, j-1$ **do**
4: Roll-out $\hat{a}_{t+i,k} = \pi_k(\hat{s}_{t+i-1,k}) \quad \forall$ agent k
5: Cross-update for each policy $k \in \{1, \ldots, K\}$
 $\hat{s}_{t+i,k} = \varphi_k\left([\hat{a}_{t+i,1}, \ldots, \hat{a}_{t+i,k}, \ldots, \hat{a}_{t+i,K}, c_{t+i}]\right)$
6: **end for**
7: Policy update for all agent k
 $\pi_k \leftarrow \texttt{Train}(\{\hat{s}_{t+i,k}, a^*_{t+i+1,k}\}_{i=0}^{j})$
8: **end for**
output K updated policies $\pi_1, \pi_2, \ldots, \pi_K$

Short-Term Memory networks (LSTM) (Hochreiter and Schmidhuber, 1997) as the policy class.[3]

Algorithm 12 outlines the `Learn` procedure for stable multi-agent imitation learning. Assume we are given consistently indexed demonstrations $A = [A_1, \ldots, A_K]$, where each $A_k = \{a_{t,k}\}_{t=1}^{T}$ corresponds action of policy π_k. Let the corresponding expert action be $a^*_{t,k}$. To lighten the notation, we denote the per-agent state vector by $s_{t,k} = \varphi_k([a_{t,1}, \ldots, a_{t,k}, \ldots, a_{t,K}, c_t])$[4]

Algorithm 12 employs a roll-out horizon j, which divides the entire trajectory into T/j segments. The following happens for every segment:

- Iteratively perform roll-out at each time step i for all K policies (line 4) to obtain actions $\{\widehat{a}_{i,k}\}$.
- Each policy simultaneously updates its state $\widehat{s}_{i,k}$, using the prediction from all other policies (line 5).
- At the end of the current segment, all policies are updated using the error signal from the deviation between predicted $\widehat{a}_{i,k}$ versus expert action $a^*_{i,k}$, for all i along the sub-segment (line 7).

[3] Note that conventional training of LSTMs does not address the cascading error problem. While LSTMs are very good at sequence-to-sequence prediction tasks, they cannot naturally deal with the drifting of input state distribution drift caused by action output feedback in dynamical systems (Bengio et al., 2015).

[4] Generally, state vector $s_{t,k}$ of policy π_k at time t can be constructed as $s_{t,k} = [\phi_k([a_{1:t,1}, c_{1:t}]), \ldots, \phi_k([a_{1:t,K}, c_{1:t}])]$

After policy updates, the training moves on to the next j-length sub-segment, using the freshly updated policies for subsequent roll-outs. The iteration proceeds until the end of the sequence is reached. In the outer loop the roll-out horizon j is incremented.

Two key insights behind our approach are:

- In addition to the training-prediction mismatch issue in single-agent learning, each agent's prediction must also be robust to imperfect predictions from other agents. This non-stationarity issue also arises in multi-agent reinforcement learning (Busoniu et al., 2008) when agents learn simultaneously. We perform joint training by cross-updating each agent's state using previous predictions from other agents.
- Many single-agent imitation learning algorithms assume the presence of a dynamic oracle to provide one-step corrections a_t^* along the roll-out trajectories. In practice, dynamic oracle feedback is very expensive to obtain and some recent work have attempted to relax this requirement (Le et al., 2016a; Ho and Ermon, 2016). Without dynamic oracles, the rolled-out trajectory can deviate significantly from demonstrated trajectories when the prediction horizon j is large ($\approx T$), leading to training instability. Thus j is gradually increased to allow for slowly learning to make good sequential predictions over longer horizons.

For efficient training, we focus on stochastic optimization, which can invoke base routine `Train` multiple times and thus naturally accommodates varying j. Note that the batch-training alternatives to Algorithm 12 can also employ similar training schemes, with similar theoretical guarantees lifted to the multi-agent case. The Appendix shows how to use DAgger (Ross et al., 2011a) for Algorithm 12, which we used for our synthetic experiment.

Coordination Structure Learning

The coordination mechanism is based on a latent structured model that governs the role assignment. The training and inference procedures seek to address two main issues:

- `LearnStructure`: unsupervised learning a probabilistic role assignment model q.

- `Assign`: how q informs the indexing mechanism so that unstructured trajectories can be mapped to structured trajectories amenable to Algorithm 12.

Given an arbitrarily ordered set of trajectories $U = \{U_1, \ldots, U_K, C\}$, let the coordination mechanism underlying each such U be governed by a true unknown model p, with global parameters θ. We suppress the agent/policy subscript and consider a generic featurized trajectory $x_t = [u_t, c_t] \ \forall t$. Let the latent role sequence for the same agent be $z = z_{1:T}$. At any time t, each agent is acting according to a latent role $z_t \sim \texttt{Categorical}\{\bar{1}, \bar{2}, \ldots, \bar{K}\}$, which are the local parameters to the structured model.

Ideally, role and index asignment can be obtained by calculating the true posterior $p(z|x, \theta)$, which is often intractable. We instead aim to approximate $p(z|x, \theta)$ by a simpler distribution q via techniques from stochastic variational inference (Hoffman et al., 2013), which allows for efficient stochastic training on mini-batches that can naturally integrate with our imitation learning subroutine.

In variational inference, posterior approximation is often cast as optimizing over a simpler model class $\mathcal{Q}$, via searching for parameters θ and z that maximize the evidence lower bound (ELBO) $\mathcal{L}$:

$$\mathcal{L}\left(q(z, \theta)\right) \triangleq \mathbb{E}_q\left[\ln p(z, \theta, x)\right] - \mathbb{E}_q\left[\ln q(z, \theta)\right] \leqslant \ln p(x)$$

Maximizing $\mathcal{L}$ is equivalent to finding $q \in \mathcal{Q}$ to minimize the KL divergence $\mathrm{KL}\left(q(z, \theta|x)||p(z, \theta|x)\right)$. We focus on the structured mean-field variational family, which factorizes q as $q(z, \theta) = q(z)q(\theta)$. This factorization breaks the dependency between θ and z, but not between single latent states z_t, unlike variational inference for i.i.d data (Kingma and Welling, 2013).

Training to learn model parameters

The procedure to learn the parameter of our structured model is summarized in Algorithm 13. Parameter learning proceeds via alternating updates over the factors $q(\theta)$ and $q(z)$, while keeping other factor fixed. Stochastic variational inference performs such updates efficiently in mini-batches. We slightly abuse notations and overload θ for the natural parameters of global parameter θ in the exponential family. Assuming the usual conjugacy in the exponential family, the stochastic natural gradient takes a convenient form (line 2 of Algo 13, and derivation in Appendix), where $t(z, x)$ is the vector of sufficient statistics, b is a vector of scaling

Algorithm 13 Structure Learning: `LearnStructure` $\{U_1, \ldots, U_K, C, \theta, \rho\} \mapsto q(\theta, z)$

Input: $X_k = \{x_{t,k}\}_{t=1}^T = \{[u_{t,k}, c_t]\}$ $\forall t, k . X = \{X_k\}_{k=1}^K$
Graphical model parameters θ, stepsize ρ

1: Local update: compute $q(z)$ via message-passing while fixing θ (See Appendix for derivations)

2: Global parameter update: via natural gradient ascent $\theta \leftarrow \theta(1-\rho) + \rho(\theta_{prior} + b^\top \mathbb{E}_{q(z)}[t(z,x)])$

output Updated model $q(\theta, z) = q(\theta)q(z)$

factors adjusting for the relative size of the mini-batches. Here the global update assumes optimal local update $q(z)$ has been computed.

Fixing the global parameters, the local updates are based on message-passing over the underlying graphical model. The exact mathematical derivation depends on the specific graph structure. The simplest scenario is to assume independence among z_t's, which resembles naive Bayes. In our experiments, we instead focus on Hidden Markov Models to capture first-order dependencies in role transitions over play sequences. In that case, line 1 of Algorithm 13 resembles running the forward-backward algorithm to compute the update $q(z)$. The forward-backward algorithm in the local update step takes $O(K^2T)$ time for a chain of length T and K hidden states. For completeness, derivation of parameter learning for HMMs is included in the Appendix.

Inference for role and index assignment

We can compute two types of inference on a learned q:

Role inference. Compute the most likely role sequence $\{z_{t,k}\}_{t=1}^T \in \{\bar{1}, \ldots, \bar{K}\}^T$, e.g., using Viterbi (or dynamic programming-based forward message passing for graph structures). This most likely role sequence for agent k, which is the low-dimensional representation of the coordination mechanism, can be used to augment the contextual feature $\{c_t\}_{t=1}^T$for each agent's policy training.

Role-based Index Assignment Transform the unstructured set U into an ordered set of trajectories A to facilitate the imitation learning step. This is the more important task for the overall approach. The intuitive goal of an indexing mechanism is to facilitate consistent agent trajectory to policy mapping. Assume for notational convenience that we want index k assigned to an unique agent who is most likely assuming role $\bar{k}$. Our inference technique rests on the well-known Linear Assignment Problem (Papadimitriou and Steiglitz, 1982), which is solved optimally via the

Algorithm 14 Multi-Agent Role Assignment: `Assign` $\{U_1, \ldots, U_K | q\} \mapsto [A_1, \ldots, A_K]$

Input: Approximate inference model q. Unordered trajectories $U = \{U_k\}_{k=1}^K$.
1: Calculate cost matrix $M \in \mathbb{R}^{K \times K}$ per equation 7.2
2: $\mathcal{A} \leftarrow$ `MinCostAssignment`(M)
output $A_k = U_{\mathcal{A}(k)} \quad \forall k = 1, 2, \ldots, K$

Kuhn-Munkres algorithm. Specifically, construct the cost matrix M as:

$$
\begin{aligned}
M &= M_1 \odot M_2 \\
M_1 &= \left[q(\{x_{t,k}\} | z_{t,k} = \bar{k}) \right] = \left[\prod_{t=1}^{T} q(x_{t,k} | z_{t,k} = \bar{k}) \right] \\
M_2 &= \left[\ln q(\{x_{t,k}\} | z_{t,k} = \bar{k}) \right] = \left[\sum_{t=1}^{T} \ln q(x_{t,k} | z_{t,k} = \bar{k}) \right]
\end{aligned}
\tag{7.2}
$$

where $k = 1, \ldots, K, \bar{k} = \bar{1}, \ldots, \bar{K}, \odot$ is the Hadamard product, and matrices M_1, M_2 take advantage of the Markov property of the graphical model. Now solving the linear assignment problem for cost matrix M, we obtain the matching $\mathcal{A}$ from role $\bar{k}$ to index k, such that the total cost per agent is minimized. From here, we rearrange the unordered set $\{U_1, \ldots, U_K\}$ to the ordered sequence $[A_1, \ldots, A_K] \equiv [U_{\mathcal{A}(1)}, \ldots, U_{\mathcal{A}(K)}]$ according to the minimum cost mapping.

To see why this index assignment procedure results in an increased entropy in the original objective (7.1), notice that:

$$
\begin{aligned}
H(\mathcal{A} | \mathcal{D}) &\approx -\sum_{\bar{k}=1}^{K} P(\bar{k}) q(\mathcal{A}(A_k) = \bar{k}) \log q(\mathcal{A}(A_k) = \bar{k}) \\
&= -\frac{1}{K} \sum_{\bar{k}=1}^{K} M(\bar{k}, k),
\end{aligned}
$$

where we assume each latent role $\bar{k}$ has equal probability. The RHS increases from the linear assignment and consequent role assignment procedure. Our inference procedure to perform role assignment is summarized in Algorithm 14.

7.4 Experiments

We present empirical results from two settings. The first is a synthetic setting based on predator-prey, where the goal is to imitate a coordinating team of predators. The second is a large-scale imitation learning setting from player trajectores in professional soccer games, where the goal is to imitate defensive team play.

Predator-Prey Domain

Setting. The predator-prey problem, also frequently called the Pursuit Domain (Benda, 1985), is a popular setting for multi-agent reinforcement learning. The traditional setup is with four predators and one prey, positioned on a grid board. At each time step, each agent has five moves: N,S,E,W or no move. The world is toroidal: the agents can move off one end of the board and come back on the other end. Agents make move simultaneously, but two agents cannot occupy the same position, and collisions are avoided by assigning a random move priority to the agents at each time step. The predators can capture the prey only if the prey is surrounded by all four predators. The goal of the predators is to capture the prey as fast as possible, which necessarily requires coordination.

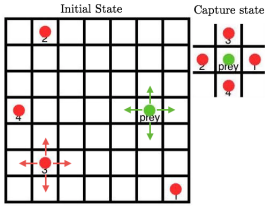


Figure 7.3:

Data. The demonstration data is collected from 1000 game instances, where four experts, indexed 1 to 4, are prescribed the consistent and coordinated role as illustrated in the capture state of Figure 7.3. In other words, agent 1 would attempt to capture the prey on the right hand side, which allows for one fixed role for each expert throughout the game. However, the particular role assignment is hidden from the imitation learning task. Each expert is then exhaustively trained using Value Iteration (Sutton and Barto, 1998) in the reinforcement learning setting, with the reward of 1 if the agent is in the position next to the prey according to its defined role, and 0 otherwise. A separate set of 100 games was collected for evaluation. A game is terminated after 50 time steps if the predators fail to capture the prey. In the test set, the experts fail to capture the prey in 2% of the games, and on average take 18.3 steps to capture the prey.

Experiment Setup. For this experiment, we use the batch version of Algorithm 11 (see appendix) to learn to imitate the experts using only demonstrations. Each policy is represented by a random forest of 20 trees, and were trained over 10 iterations. The expert correction for each rolled-out state is collected via Value Iteration. The experts thus act as dynamic oracles, which result in a multi-agent training setting analogous to DAgger (Ross et al., 2011a). We compare two versions of multi-agent imitation learning:

- **Coordinated Training.** We use our algorithm, with the latent structure model represented by a discrete Hidden Markov Model with binomial emission. We

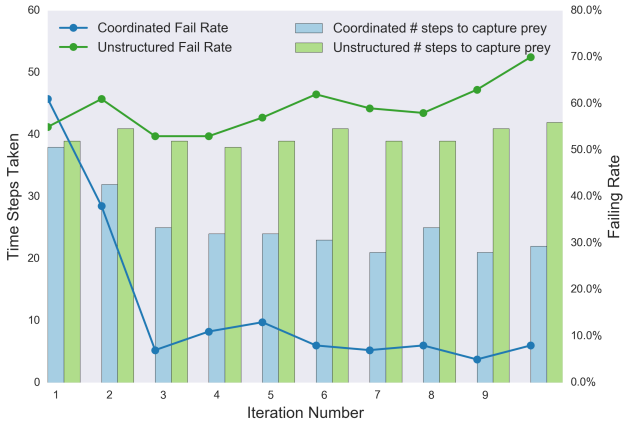


Figure 7.4: *Comparing performance in Predator-Prey between our approach and unstructured multi-agent imitation learning, as a function of the number of training rounds. Our approach demonstrates both significantly lower failure rates as well as lower average time to success (for successful trials).*

use Algorithm 14 to maximize the role consistency of the dynamic oracles across different games.

- **Unstructured Training.** An arbitrary role is assigned to each dynamic oracle for each game, i.e., the agent index is meaningless.

In both versions, training was done using the same data aggregation scheme and batch training was conducted using the same random forests configuration.

Results. Figure 7.4 compares the test performance of our method versus unstructured multi-agent imitation learning. Our method quickly approaches expert performance (average 22 steps with 8% failure rate in the last iteration), whereas unstructured multi-agent imitation learning performance did not improve beyond the first iteration (average 42 steps with 70% failure rate). Note that we even gave the unstructured baseline some advantage over our method, by forcing the prey to select the moves last after all predators make decisions (effectively making the prey slower). Without this advantage, the unstructured policies fail to capture the prey almost 100% of the time. Also, if the same restriction is applied to the policies obtained from our method, performance would be on par with the experts (100% success rate, with similar number of steps taken).

Multi-agent Imitation Learning for Soccer

Setting. Soccer is a popular domain for multi-agent learning. RoboCup, the robotic and simulation soccer platform, is perhaps the most popular testbed for multi-agent

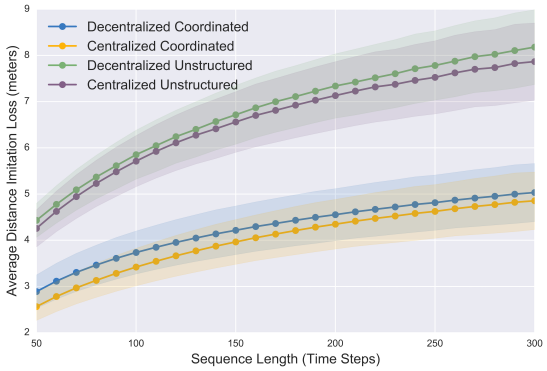


Figure 7.5: *Experimental results on soccer domain. We see that using coordination substantially improves the imitation loss, and that the decentralized policy is comparable to the centralized.*

reinforcement learning research to date (Stone, 2016). The success of MARL has been limited, however, due to the extremely high dimensionality of the problem. In this experiment, we aim to learn multi-agent policies for team soccer defense, based on tracking data from real-life professional soccer (Bialkowski et al., 2014).

Data. We use the tracking data from 45 games of real professional soccer from a recent European league. The data was chunked into sequences with one team attacking and the other defending. Our goal is to learn up to 10 policies for team defense (11 players per team, minus the goal keeper). The training data consists of 7500 sets of trajectories $A = \{A_1, \ldots, A_{10}\}$, where $A_k = \{a_{t,k}\}_{t=1}^{T}$ is the sequence of positions of one defensive player, and C is the context consisting of opponents and the ball. Overall, there are about 1.3 million frames at 10 frames per second. The average sequence length is 176 steps, and the maximum is 1480.

Experiment Setup. Each policy is represented by a recurrent neural network structure (LSTM), with two hidden layers of 512 units each. As LSTMs generally require fixed-length input sequences, we further chunk each trajectory into subsequences of length 50, with overlapping window of 25 time steps. The joint multi-agent imitation learning procedure follows Algorithm 12 closely. In this setup, without access to dynamic oracles for imitation learning in the style of SEARN (Daumé III et al., 2009) and DAgger (Ross et al., 2011a), we gradually increase the horizon of the rolled-out trajectories from 1 to 10 steps look-ahead. Empirically, this has the effect of stabilizing the policy networks early in training, and limits the cascading errors caused by rolling-out to longer horizons.

The structured model component is learned via stochastic variational inference on a continuous HMM, where the per-state emission distribution is a mixture of

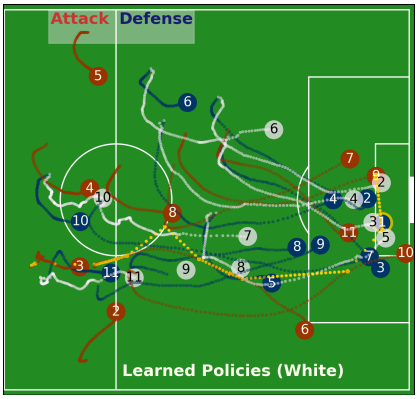


Figure 7.6: *Result of 10 coordinated imitation policies, corresponding with Figure 7.1. White is the rolled-out imitation policies.*

Gaussians. Training and inference operate on the same mini-batches used for joint policy learning.

We compare against two variations. The first employs centralized policy that aggregates the state vectors of all decentralized learner and produces the actions for all players, i.e., a multi-task policy. The centralized approach generally requires more model parameters, but is potentially much more accurate. The second variation is to not employ joint multi-agent training: we modify Algorithm 12 to not cross-update states between agents, and each role is trained conditioned on the ground truth of the other agents.

Results. Figure 7.5 shows the results. Our coordinated learning approach substantially outperforms conventional imitation learning without structured coordination. The imitation loss measures average distance of roll-outs and ground truth in meters (note the typical size of soccer field is 110×70 meters). As expected, average loss increases with longer sequences, due to cascading errors. However, this error scales sub-linearly with the length of the horizon, even though the policies were trained on sequences of length 50. Note also that the performance difference between decentralized and centralized policies is insignificant compared to the gap between coordinated and unstructured policies, further highlighting the benefits of structured coordination in multi-agent settings. The loss of a single network, non-joint training scheme is very large and thus omitted from Figure 7.5 (see the appendix).

Visualizations. Imitation loss, of course, is not a full reflection of the quality of the learned policies. Unlike predator-prey, the long-term reward signal is not available, so we rely on visual inspection as part of evaluation. Figure 7.6 overlays policy prediction on top of the actual game sequence from Figure 7.1. Additional

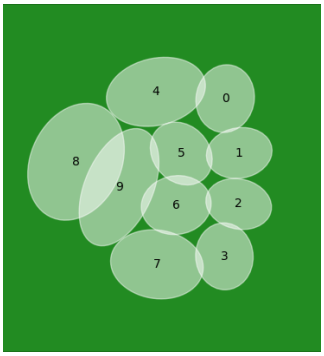


Figure 7.7: *Components of role distributions, corresponding to a popular formation arrangement in professional soccer*

test examples are included in supplemental videos [5]. We note that learned policies are qualitatively similar to the ground truth demonstrations, and can be useful for applications such as counterfactual analysis (Le et al., 2017). Figure 7.7 displays the Gaussian components of the underlying HMM. The components correspond to the dominant modes of the roles assigned. Unlike the predator-prey domain, roles can be switched during a sequence of play. See the appendix for more details on role swap frequency.

7.5 Related Work in Multi-Agent Learning Context

The problem of multi-agent imitation learning has not been widely considered, perhaps with the exception of (Chernova and Veloso, 2007) which focused on very different applications and technical challenges (i.e., learning a model of a joint task by collecting samples from direct interaction with teleoperating human teachers). The actual learning algorithm there requires the learner to collect enough data points from human teachers for confident classification of task. It is not clear how well the proposed method would translate to other domains.

Index-free policy learning is generally difficult for black-box machine learning techniques. Some recent work has called attention to the importance of order to learning when input or output are sets (Vinyals et al., 2015), motivated by classic algorithmic and geometric problems such as learning to sort a set of numbers, or finding convex hull for a set of points, where no clear indexing mechanism exists. Other permutation invariant approaches include those for standard classification (Shivaswamy and Jebara, 2006).

[5] Additional videos can be seen at `http://hoangle.info`

7.6 Limitations and Discussions

In principle, the training and inference of the latent structure model can accommodate different types of graphical models. However, the exact procedure varies depending on the graph structure. It would be interesting to find domains that can benefit from more general graphical models. Another possible direction is to develop fully end-to-end differentiable training methods that can accommodate our index-free policy learning formulation, especially deep learning-based method that could provide computational speed-up compared to traditional graphical model inference. One potential issue with the end-to-end approach is the need to depart from a learning-reductions style approach.

Although we addressed learning from demonstrations in this chapter, the proposed framework can also be employed for generative modeling, or more efficient structured exploration for reinforcement learning. Along that line, our proposed method could serve as a useful component of general reinforcement learning, especially in multi-agent settings where traditional exploration-based approaches such as Q-learning prove computationally intractable.

Chapter 8

CONCLUDING REMARKS

In this dissertation, we have developed three groups of techniques for *structured policy learning*. To summarize, in Chapter 2 and 3, we discussed value-based structured policy learning, which imposes constraints on overall performance of the policy as an external specification. In Chapter 4 and 5, we showed that building structural constraints directly into the policy class can yield both certifiable policy performance, as well as improved data efficiency in the setting of constrained imitation learning. In Chapter 6 and 7, we demonstrated that exploiting partial knowledge of the model can significantly improve the data efficiency of learning, as well as alleviating the generally difficult imperfect observation scenarios.

As an overarching message, policy learning has much to gain from bringing systematic structure into learning. Not only can we achieve more data-efficient learning, but the learning agent can also better capture what we ultimately care about in real-life applications. Through our applications, structured policy learning has shown encouraging performance, and is an overall more reasonable approach compared to the alternative paradigm of end-to-end learning, which has been powered by the recent advances in deep learning.

Looking forward, we do have several interesting questions that arise from the approach of this thesis:

- How do we reason about generalization power of policy learning when we incorporate structural constraints? How do we develop a further unifying perspectives to bring together different types of structural constraints presented in this work? An interesting research direction as a proof of concept would be bringing value-based and policy-based approaches together, or alternatively, indentifying methods to unify different model-based structure into one common framework for better analysis and algorithm design

- In addition to methodological advances, there is perhaps no better replacement for real-world policy learning than trying to solve real-world problems. The research community would benefit from expanding into better benchmarking testbeds that reflect realistic constraints. More broadly, this opens up the

question of how do we interface policy learning methods with established methods from other disciplines. One of the emerging lessons from our recent work is that we have much to learn from the algorithmic techniques that other disciplines have come up with for decision making, such as game theory, or program synthesis, or control theory and others. Such a fusion of knowledge can potentially accelerate the advent of a learning-based systems of the future that best serve the common interests

BIBLIOGRAPHY

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.

Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2005.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31, 2017.

Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*, pages 1638–1646, 2014.

Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *International Conference on Machine Learning*, 2018.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pages 1–8. IEEE, 2013.

Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. Scaling enumerative program synthesis via divide and conquer. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, pages 319–336, 2017.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Kiam Heong Ang, Gregory Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE transactions on control systems technology*, 13(4):559–576, 2005.

András Antos, Csaba Szepesvári, and Rémi Munos. Fitted q-iteration in continuous action-space mdps. In *Advances in neural information processing systems*, pages 9–16, 2008a.

András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008b.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5): 469–483, 2009.

Karl Johan Åström and Tore Hägglund. Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, 20(5):645–651, 1984.

J Andrew Bagnell, Sham M Kakade, Jeff G Schneider, and Andrew Y Ng. Policy search by dynamic programming. In *Neural Information Processing Systems (NIPS)*, 2003.

Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Heejung Bang and James M. Robins. Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61(4):962–973, 2005. doi: 10.1111/j.1541-0420.2005.00377.x.

Peter L Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension bounds for piecewise linear neural networks. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT 2017)*, 2017.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

Heinz H Bauschke, Patrick L Combettes, et al. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer, 2011.

Matthew James Beal. *Variational algorithms for approximate Bayesian inference*. University of London United Kingdom, 2003.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Richard Bellman, Irving Glicksberg, and Oliver Gross. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 14(1):11–18, 1956.

Miroslav Benda. On optimal cooperation of knowledge sources. *Technical Report BCS-G2010-28*, 1985.

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

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 2005.

Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, and Iain Matthews. "win at home and draw away": Automatic formation analysis highlighting the differences in home and away team behaviors. In *MIT Sloan Sports Analytics Conference (SSAC)*, 2014.

Lars Blackmore, Masahiro Ono, and Brian C Williams. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, 2011.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112 (518):859–877, 2017.

Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 140–156, 2009.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Léon Bottou, Jonas Peters, Joaquin Qui nonero Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research (JMLR)*, 14:3207–3260, 2013.

Philippe Bougerol and Nico Picard. Strict stationarity of generalized autoregressive processes. *The Annals of Probability*, pages 1714–1730, 1992.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

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

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *International Conference on Machine Learning (ICML)*, 2006.

Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103, 2008.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume III, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. Bridging boolean and quantitative synthesis using smoothed proof search. In *POPL*, pages 207–220, 2014.

Jianhui Chen and Peter Carr. Mimicking human camera operators. In *IEEE Winter Conference Applications of Computer Vision (WACV)*, 2015.

Jianhui Chen, Hoang M. Le, Peter Carr, Yisong Yue, and James J. Little. Learning online smooth predictors for real-time camera planning using recurrent decision trees. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2019.

C Cheng, X Yan, N Wagener, and B Boots. Fast policy learning through imitation and reinforcement. In *Uncertainty in artificial intelligence*, 2019a.

Ching-An Cheng, Xinyan Yan, Nathan Ratliff, and Byron Boots. Predictor-corrector policy optimization. In *International Conference on Machine Learning*, pages 1151–1161, 2019b.

Ching-An Cheng, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Accelerating imitation learning with predictive models. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019c.

Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick. Control regularization for reduced variance reinforcement learning. In *International Conference on Machine Learning*, pages 1141–1150, 2019d.

Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick. Control regularization for reduced variance reinforcement learning. In *International Conference on Machine Learning*, pages 1141–1150, 2019e.

Sonia Chernova and Manuela Veloso. Multiagent collaborative task learning through imitation. In *Proceedings of the fourth International Symposium on Imitation in Animals and Artifacts*, pages 74–79, 2007.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *NIPS*, 2017.

Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2–3):81–227, 2012.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *NIPS*, 1993.

Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, pages 337–340, 2008.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 179–186. Omnipress, 2012.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 990–998. JMLR. org, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13(1):227–303, 2000.

Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient rl with rich observations via latent state decoding. In *International Conference on Machine Learning*, pages 1665–1674, 2019.

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: automatic conversion of 3d models to CSG trees. *ACM Trans. Graph.*, 37(6):213:1–213:16, 2018.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

John C Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *COLT*, pages 14–26, 2010.

Miroslav Dudik, Steven J Phillips, and Robert E Schapire. Performance guarantees for regularized maximum entropy density estimation. In *International Conference on Computational Learning Theory*, pages 472–486. Springer, 2004.

Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. In *International Conference on Machine Learning (ICML)*, 2011a.

Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 1097–1104. Omnipress, 2011b.

Layla El Asri, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman. Frames: a corpus for adding memory to goal-oriented dialogue systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 207–219, 2017.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems*, pages 6059–6068, 2018.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Amir M Farahmand, Mohammad Ghavamzadeh, Shie Mannor, and Csaba Szepesvári. Regularized policy iteration. In *Advances in Neural Information Processing Systems*, pages 441–448, 2009.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. *arXiv preprint arXiv:1802.03493*, 2018a.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *International Conference on Machine Learning (ICML)*, 2018b.

John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 229–239, 2015.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 385–394. Society for Industrial and Applied Mathematics, 2005.

Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Springer, 2001.

Ronan Fruit and Alessandro Lazaric. Exploration–exploitation in mdps with options. *arXiv preprint arXiv:1703.08667*, 2017.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine learning*, 89(1-2):123–156, 2012.

Vamsidhar Reddy Gaddam, Ragnhild Eg, Ragnar Langseth, Carsten Griwodz, and Pål Halvorsen. The cameraman operating my virtual camera is artificial: Can the machine be as good as a human&quest. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(4):56, 2015.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Yves Grandvalet, Yoshua Bengio, et al. Semi-supervised learning by entropy minimization. In *NIPS*, volume 17, pages 529–536, 2004.

Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234, 2002.

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

Zhaohan Guo, Philip S Thomas, and Emma Brunskill. Using options and covariance testing for long horizon off-policy policy evaluation. In *Advances in Neural Information Processing Systems*, pages 2492–2501, 2017.

László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A distribution-free theory of nonparametric regression*. Springer Science & Business Media, 2006.

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

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361, 2017.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774, 2017.

John Michael Hammersley and David Christopher Handscomb. Monte carlo methods. 1964.

Josiah Hanna, Scott Niekum, and Peter Stone. Importance sampling policy evaluation with an estimated behavior policy. In *International Conference on Machine Learning (ICML)*, 2019.

Anna Harutyunyan, Marc G. Bellemare, Tom Stepleton, and Rémi Munos. Q(lambda) with off-policy corrections. In *International Conference on Algorithmic Learning Theory (ALT)*, 2016.

Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *ICLR*, 2016.

David Haussler. Sphere packing numbers for subsets of the boolean n-cube with bounded vapnik-chervonenkis dimension. *Journal of Combinatorial Theory, Series A*, 69(2):217–232, 1995.

He He, Jason Eisner, and Hal Daume. Imitation learning by coaching. In *Neural Information Processing Systems (NIPS)*, 2012.

Ruijie He, Emma Brunskill, and Nicholas Roy. Puma: Planning under uncertainty with macro-actions. In *AAAI*, 2010.

Mikael Henaff, Alfredo Canziani, and Yann LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. *arXiv preprint arXiv:1901.02705*, 2019.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

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

Matt Hoffman and David Blei. Structured stochastic variational inference. *CoRR abs/1404.4114*, 2014.

Matthew D Hoffman, David M Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1): 1303–1347, 2013.

Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Neural Information Processing Systems (NIPS)*, 2013.

Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 215–224. ACM, 2010.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661, 2016a.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016b.

Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low bellman rank are pac-learnable. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1704–1713. JMLR. org, 2017.

Matthew James Johnson and Alan S Willsky. Stochastic variational inference for bayesian time series models. In *ICML*, pages 1854–1862, 2014.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2002.

Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

Nathan Kallus and Masatoshi Uehara. Double reinforcement learning for efficient off-policy evaluation in markov decision processes. *arXiv preprint arXiv:1908.08526*, 2019a.

Nathan Kallus and Masatoshi Uehara. Efficiently breaking the curse of horizon: Double reinforcement learning in infinite-horizon processes. *arXiv preprint arXiv:1909.05850*, 2019b.

Nathan Kallus and Angela Zhou. Policy evaluation and optimization with continuous treatments. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

Joseph DY Kang, Joseph L Schafer, et al. Demystifying double robustness: A comparison of alternative strategies for estimating a population mean from incomplete data. *Statistical science*, 22(4):523–539, 2007.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Peter Kingston and Magnus Egerstedt. Index-free multi-agent systems: An eulerian approach. *IFAC Proceedings Volumes*, 43(19):215–220, 2010.

Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

Stephen Kloder and Seth Hutchinson. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*, 22(4):650–665, 2006.

Sven Koenig and Reid G Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22(1-3):227–250, 1996.

Jelle R Kok, Matthijs TJ Spaan, Nikos Vlassis, et al. Multi-robot decision making using coordination graphs. In *Proceedings of the 11th International Conference on Advanced Robotics, ICAR*, volume 3, pages 1124–1129, 2003.

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

Andreas Krause, Pietro Perona, and Ryan G Gomes. Discriminative clustering by regularized information maximization. In *Advances in neural information processing systems*, pages 775–783, 2010.

Akshay Krishnamurthy, Alekh Agarwal, and John Langford. Pac reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, pages 1840–1848, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, pages 3675–3683, 2016.

Michail Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *International Conference on Machine Learning (ICML)*, 2003a.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003b.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

John Langford and Bianca Zadrozny. Relating reinforcement learning performance to classification performance. In *International Conference on Machine Learning (ICML)*, 2005.

Alessandro Lazaric and Marcello Restelli. Transfer from multiple mdps. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2011.

Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Finite-sample analysis of lstd. In *ICML-27th International Conference on Machine Learning*, pages 615–622, 2010.

Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13(Oct):3041–3074, 2012.

Hoang Le, Andrew Kang, Yisong Yue, and Peter Carr. Smooth imitation learning for online sequence prediction. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 680–688, 2016a.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712, 2019a.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712, 2019b.

Hoang M. Le, Andrew Kang, Yisong Yue, and Peter Carr. Smooth imitation learning for online sequence prediction. In *International Conference on Machine Learning (ICML)*, 2016b.

Hoang M. Le, Peter Carr, Yisong Yue, and Patrick Lucey. Data-driven ghosting using deep imitation learning. In *MIT Sloan Sports Analytics Conference (SSAC)*, 2017.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Wee Sun Lee, Peter L Bartlett, and Robert C Williamson. Efficient agnostic learning of neural networks with bounded fan-in. *IEEE Transactions on Information Theory*, 42(6):2118–2132, 1996.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *International Conference on Web Search and Data Mining (WSDM)*, 2011.

Lihong Li, Remi Munos, and Csaba Szepesvari. Toward minimax off-policy value estimation. In *Artificial Intelligence and Statistics*, pages 608–616, 2015.

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

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Lydia Liu, Sarah Dean, Esther Rolf, Max Simchowitz, and Moritz Hardt. Delayed impact of fair machine learning. In *International Conference on Machine Learning*, pages 3156–3164, 2018a.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Neural Information Processing Systems (NIPS)*, 2018b.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Advances in Neural Information Processing Systems*, pages 5361–5371, 2018c.

Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *arXiv preprint arXiv:1904.08473*, 2019.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *International Conference on Machine Learning*, pages 4114–4124, 2019.

Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. In *AAMAS*, 2016.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Ji Yang, Minmin Chen, Jiaxi Tang, Lichan Hong, and Ed H Chi. Off-policy learning in two-stage recommender systems. In *Proceedings of The Web Conference 2020*, pages 463–473, 2020.

Sridhar Mahadevan and Bo Liu. Sparse q-learning with mirror descent. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 564–573. AUAI Press, 2012.

Odalric-Ambrym Maillard, Rémi Munos, Alessandro Lazaric, and Mohammad Ghavamzadeh. Finite-sample analysis of bellman residual minimization. In *Proceedings of 2nd Asian Conference on Machine Learning*, pages 299–314, 2010.

Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus Frean. Functional gradient techniques for combining hypotheses. In *Neural Information Processing Systems (NIPS)*, 1999.

Dipendra Misra, Mikael Henaff, Akshay Krishnamurthy, and John Langford. Kinematic state abstraction and provably efficient rich-observation reinforcement learning. *arXiv preprint arXiv:1911.05815*, 2019.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

William H Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

Rémi Munos. Error bounds for approximate policy iteration. In *ICML*, volume 3, pages 560–567, 2003.

Rémi Munos. Performance bounds in l_p-norm for approximate value iteration. *SIAM journal on control and optimization*, 46(2):541–561, 2007.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(May):815–857, 2008.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

Vijayaraghavan Murali, Swarat Chaudhuri, and Chris Jermaine. Neural sketch learning for conditional program generation. In *ICLR*, 2018.

Susan A Murphy, Mark J van der Laan, James M Robins, and Conduct Problems Prevention Research Group. Marginal mean models for dynamic regimes. *Journal of the American Statistical Association*, 96(456):1410–1423, 2001.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. In *Advances in Neural Information Processing Systems*, pages 2315–2325, 2019.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *ICRA*, 2017.

Arkadii Semenovich Nemirovsky and David Borisovich Yudin. *Problem complexity and method efficiency in optimization.* Wiley, 1983.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

Xinkun Nie, Emma Brunskill, and Stefan Wager. Learning when-to-treat policies. *arXiv preprint arXiv:1905.09751*, 2019.

Michael Oberst and David Sontag. Counterfactual off-policy evaluation with gumbel-max structural causal models. In *International Conference on Machine Learning*, pages 4881–4890, 2019.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning*, 2018.

Masahiro Ono, Marco Pavone, Yoshiaki Kuwata, and J Balaram. Chance-constrained dynamic programming with application to risk-aware robotic space exploration. *Autonomous Robots*, 39(4):555–571, 2015.

Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2-3):161–178, 2002.

Cosmin Paduraru. *Off-policy evaluation in Markov decision processes*. PhD thesis, McGill University Libraries, 2013.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntak Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. In *Robotics: science and systems*, 2018.

Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2231–2240, 2017.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, and Hervé Frezza-Buet. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)*, 7 (3):7, 2011.

Oleksandr Polozov and Sumit Gulwani. Flashmeta: a framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pages 107–126, 2015.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

Michael JD Powell and J Swann. Weighted uniform sampling—a monte carlo technique for reducing variance. *IMA Journal of Applied Mathematics*, 2(3): 228–236, 1966.

Niranjani Prasad, Li-Fang Cheng, Corey Chivers, Michael Draugelis, and Barbara E Engelhardt. A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. *arXiv preprint arXiv:1704.06300*, 2017.

Doina Precup, Richard S Sutton, and Satinder P Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 759–766. Morgan Kaufmann Publishers Inc., 2000.

Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal difference learning with function approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 417–424. Morgan Kaufmann Publishers Inc., 2001.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

Nathan Ratliff, David Silver, and J. Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

Veselin Raychev, Pavol Bielik, Martin T. Vechev, and Andreas Krause. Learning programs from noisy data. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 761–774, 2016.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stephane Ross, Geoff Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011a.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, pages 627–635, 2011b.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015a.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Statist.*, 21 (1):124–127, 03 1950. doi: 10.1214/aoms/1177729893.

Pannagadatta K Shivaswamy and Tony Jebara. Permutation invariant svms. In *Proceedings of the 23rd international conference on Machine learning*, pages 817–824. ACM, 2006.

Xujie Si, Yuan Yang, Hanjun Dai, Mayur Naik, and Le Song. Learning a meta-solver for syntax-guided program synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

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

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354, 2017.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415, 2006.

Jonathan Sorg, Richard L Lewis, and Satinder P Singh. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems*, pages 2190–2198, 2010.

Nathan Srebro, Karthik Sridharan, and Ambuj Tewari. Smoothness, low noise and fast rates. In *Neural Information Processing Systems (NIPS)*, 2010.

Peter Stone. What's hot at RoboCup. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017.

Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. In *International Conference on Learning Representations (ICLR)*, 2018a.

Wen Sun, Geoffrey J Gordon, Byron Boots, and J Bagnell. Dual policy iteration. In *Advances in Neural Information Processing Systems*, pages 7059–7069, 2018b.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018a.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018b.

Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. In *ICML*, volume 98, pages 556–564, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

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

Adith Swaminathan and Thorsten Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research*, 16(1):1731–1755, 2015.

Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*, pages 3632–3642, 2017.

Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *NIPS*, pages 1449–1456, 2008.

Umar Syed and Robert E Schapire. A reduction from apprenticeship learning to classification. In *Neural Information Processing Systems (NIPS)*, 2010.

Sarah Taylor, Taehwan Kim, Yisong Yue, Moshe Mahler, James Krahe, Anastasio Garcia Rodriguez, Jessica Hodgins, and Iain Matthews. A deep learning approach for generalized speech animation. *ACM Transactions on Graphics (TOG)*, 36(4):93, 2017.

Guy Tennenholtz, Shie Mannor, and Uri Shalit. Off-policy evaluation in partially observable environments. *arXiv preprint arXiv:1909.03739*, 2019.

Philip Thomas. Reinforcement learning: An introduction, 2015.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016a.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016b.

Philip S Thomas, William C Dabney, Stephen Giguere, and Sridhar Mahadevan. Projected natural actor-critic. In *Advances in neural information processing systems*, pages 2337–2345, 2013.

Philip S Thomas, Georgios Theocharous, Mohammad Ghavamzadeh, Ishan Durugkar, and Emma Brunskill. Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. 2017.

Masatoshi Uehara and Nan Jiang. Minimax weight and q-function learning for off-policy evaluation. *arXiv preprint arXiv:1910.12809*, 2019.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pages 8687–8698, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5052–5061, 2018.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind Blog*, 2019.

John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007.

Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudík. Optimal and adaptive off-policy evaluation in contextual bandits. In *International Conference on Machine Learning*, pages 3589–3597, 2017.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, pages 1995–2003, 2016.

MA Wiering. Multi-agent reinforcement learning for traffic light control. In *International Conference on Machine Learning (ICML)*, 2000.

Herman Wold. A study in the analysis of stationary time series, 1939.

Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, The Open Racing Car Simulator. http://www.torcs.org, 2014.

Tengyang Xie, Yifei Ma, and Yu-Xiang Wang. Towards optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. In *Neural Information Processing Systems (NIPS)*, 2019.

Shangtong Zhang, Bo Liu, and Shimon Whiteson. Gradientdice: Rethinking generalized offline estimation of stationary values. *arXiv preprint arXiv:2001.11113*, 2020.

Stephan Zheng, Yisong Yue, and Patrick Lucey. Generating long-term trajectories using deep hierarchical networks. In *NIPS*, 2016.

He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *ACM Conference on Programming Language Design and Implementation (SIGPLAN)*, 2019.

Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, CMU, 2010.

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

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.

Appendix A

APPENDIX TO CHAPTER 2

In this appendix section, we provide additional discussion and detailed proofs for the results presented in Chapter 2.

A.1 Equivalence between Regularization and Constraint Satisfaction

Formulating Different Regularized Policy Learning Problems as Constrained Policy Learning

In this section, we provide connections between regularized policy learning and our constrained formulation (OPT). Although the main chapter in the thesis focuses on batch policy learning, here we are agnostic between online and batch learning settings.

Entropy regularized RL. The standard reinforcement learning objective, either in online or batch setting, is to find a policy π^*_{std} that minimizes the long-term cost (equivalent to maximizing the accumuted rewards):

$$\pi^*_{\text{std}} = \arg\min_{\pi} \sum_t \mathbb{E}_{(x_t,a_t)\sim\pi}[c(x_t,a_t)] = \arg\min_{\pi} \mathbb{E}_{(x,a)\sim\mu_\pi}[c(x,a)]$$

Maximum entropy reinforcement learning (Haarnoja et al., 2017) augments the cost with an entropy term, such that the optimal policy maximizes its entropy at each visited state: $\pi^*_{\text{MaxEnt}} = \arg\min_\pi \mathbb{E}_{(x,a)\sim\mu_\pi}[c(x,a)] - \lambda\mathbb{H}(\pi(\cdot|x))$. As discuseed by (Haarnoja et al., 2017), the goal is for the agent to maximize the entropy of the entire trajectory, and not greedily maximizing entropy at the current time step (i.e., Boltzmann exploration). Maximum entropy policy learning was first proposed by (Ziebart et al., 2008; Ziebart, 2010) in the context of learning from expert demonstrations. Entropy regulazed RL/IL is equivalent to our problem (OPT) by simply set $C(\pi) = \mathbb{E}_{(x_t,a_t)\sim\pi}[c(x_t,a_t)]$ (standard RL objective), and $g(x,a) = \pi(a|x)\log\pi(a|x)$, thus $G(\pi) = -\mathbb{H}(\pi) \leqslant \tau$

Smooth imitation learning (& Regularized imitation learning). This is a constrained imitation learning problem studied by (Le et al., 2016a): learning to mimic smooth behavior in continuous space from human desmonstrations. The data collected from human demonstrations is considered to be fixed and given a priori, thus the imitation learning task is also a batch policy learning problem.

The proposed approach from (Le et al., 2016a) is to view policy learning as a function regularization problem: policy $\pi = (f, g)$ is a combination of functions f and h, where f belongs to some expressive function class F (e.g., decision trees, neural networks) and $h \in$ H with certifiable smoothness property (e.g., linear models). Policy learning is the solution to the functional regularization problem $\pi = \arg\min_{f,g} \mathbb{E}_{x\sim\mu_\pi} \|f(x) - \pi_E(x)\| + \lambda \|h(x) - \pi_E(x)\|$, where π_E is the expert policy. This constrained imitation learning setting is equivalent to our problem (OPT) as follows: $C(\pi) = C((f,h)) = \mathbb{E}_{x\sim\mu_\pi} \|f(x) - \pi_E(x)\|$ and $G(\pi) = G((f,h)) = \min_{h'\in \mathrm{H}} \|h'(x) - \pi_E(x)\| \leqslant \tau$

Regularizing RL with expert demonstrations / Learning from imperfect demonstrations. Efficient exploration in RL is a well-known challenge. Expert demonstrations provide a way around online exploration to reduce the sample complexity for learning. However, the label budget for expert demonstrations may be limited, resulting in a sparse coverage of the state space compared to what the online RL agent can explore (Hester et al., 2018). Additionally, expert demonstrations may be imperfect (Oh et al., 2018). Some recent work proposed to regularize standard RL objective with some deviation measure between the learning policy and (sparse) expert data (Hester et al., 2018; Oh et al., 2018; Henaff et al., 2019).

For clarity we focus on the regularized RL objective for Q-learning in (Hester et al., 2018), which is defined as $J(\pi) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)$, where $J_{DQ}(Q)$ is the standard deep Q-learning loss, $J_n(Q)$ is the n-step return loss, $J_E(Q)$ is the imitation learning loss defined as $J_E(Q) = \max_{a\in \mathrm{A}} [Q(x,a) + \ell(a_E, a) - Q(x, a_E)]$, and $J_{L2}(Q)$ is an L2 regularization loss applied to the Q-network to prevent overfitting to a small expert dataset. The regularization parameters λ's are obtained by hyperparameter tuning. This approach provides a bridge between RL and IL, whose objective functions are fundamentally different (see AggreVate by (Ross and Bagnell, 2014) for an alternative approach).

We can cast this problem into (OPT) as: $C(\pi) = C_{DQ}(Q) + \lambda_3 C_{L2}(Q)$ (standard RL objective), and two constraints: $g_1(\pi) = \mathbb{E}_{x\sim\mu_\pi}[\max_{a\in \mathrm{A}} Q(x,a) + \ell(a_E, a) - Q(x, a_E)]$, and
$g_2(x,a) = \mathbb{E}_{x\sim\mu_\pi}[c_t + \gamma c_{t+1} + \ldots + \gamma^{n-1} c_{t+n-1} + \min'_a \gamma^n Q(x_{t+n}, a') - Q(x_t, a)]$
Here g_1 captures the loss w.r.t. expert demonstrations and g_2 reflects the n-step return constraint.

More generally, one can define the imitation learning constraint as
$G(\pi) = \mathbb{E}_{x\sim\mu_\pi} \ell(\pi(x), \pi_E(x))$ for an appropriate divergence definition between $\pi(x)$

and $\pi_E(x)$ (defined at states where expert demonstrations are available).

Conservative policy improvement. Many policy search algorithms perform small policy update steps, requiring the new policy π to stay within a neighborhood of the most recent policy iterate π_k to ensure learning stability (Levine and Abbeel, 2014; Schulman et al., 2015; Montgomery and Levine, 2016; Achiam et al., 2017). This simply corresponds to the definition of $G(\pi) = \texttt{distance}(\pi, \pi_k) \leqslant \tau$, where `distance` is typically KL-divergence or total variation distance between the distribution induced by π and π_k. For KL-divergence, the single timestep cost $g(x,a) = -\pi(a|x)\log(\frac{\pi_k(a|x)}{\pi(a|x)})$

Equivalence of Regularization and Constraint Viewpoint - Proof of Proposition 2.2.1

`Regularization` $\Longrightarrow$ `Constraint`: Let $\lambda > 0$ and π^* be optimal policy in `Regularization`. Set $\tau = G(\pi^*)$. Suppose that π^* is not optimal in `Constraint`. Then $\exists \pi \in \Pi$ such that $G(\pi) \leqslant \tau$ and $C(\pi) < C(\pi^*)$. We then have

$$C(\pi) + \lambda^\top G(\pi) < C(\pi^*) + \lambda^\top \tau = C(\pi^*) + \lambda^\top G(\pi^*)$$

which contradicts the optimality of π^* for `Regularization` problem. Thus π^* is also the optimal solution of the `Constraint` problem.

`Constraint` $\Longrightarrow$ `Regularization`: Given τ and let π^* be the corresponding optimal solution of the `Constraint` problem. The Lagrangian of `Constraint` is given by $L(\pi,\lambda) = C(\pi) + \lambda^\top G(\pi), \lambda \geqslant 0$. We then have $\pi^* = \arg\min_{\pi\in\Pi} \max_{\lambda \geqslant 0} L(\pi,\lambda)$. Let

$$\lambda^* = \arg\max_{\lambda \geqslant 0} \min_{\pi\in\Pi} L(\pi,\lambda)$$

Slater's condition implies strong duality. By strong duality and the strong max-min property (Boyd and Vandenberghe, 2004), we can exchange the order of maximization and minimization. Thus π^* is the optimal solution of

$$\min_{\pi\in\Pi} \quad C(\pi) + (\lambda^*)^\top (G(\pi) - \tau)$$

Removing the constaint $(\lambda^*)^\top \tau$, we have that π^* is the optimal solution of the `Regularization` problem with $\lambda = \lambda^*$. And since $\pi^* \neq \arg\min_{\pi\in\Pi} C(\pi)$, we must have $\lambda^* \geqslant 0$.

A.2 Convergence Proofs

Convergence of Meta-algorithm - Proof of Proposition 2.3.1

Let us evaluate the empirical primal-dual gap of the Lagrangian after T iterations:

$$\max_{\lambda} L(\hat{\pi}_T, \lambda) = \max_{\lambda} \frac{1}{T} \sum_t L(\pi_t, \lambda) \tag{A.1}$$

$$\leqslant \frac{1}{T} \sum_t L(\pi_t, \lambda_t) + \frac{o(T)}{T} \tag{A.2}$$

$$\leqslant \frac{1}{T} \sum_t L(\pi, \lambda_t) + \frac{o(T)}{T} \quad \forall \pi \in \Pi \tag{A.3}$$

$$= L(\pi, \widehat{\lambda}_T) + \frac{o(T)}{T} \quad \forall \pi \tag{A.4}$$

Equations (A.1) and (A.4) are due to the definition of $\hat{\pi}_T$ and $\widehat{\lambda}_T$ and linearity of $L(\pi, \lambda)$ wrt λ and the distribution over policies in Π. Equation (A.2) is due to the no-regret property of `Online-algorithm`. Equation (A.3) is true since π_t is best response wrt λ_t. Since equation (A.4) holds for all π, we can conclude that for T sufficiently large such that $\frac{o(T)}{T} \leqslant \omega$, we have $\max_\lambda L(\hat{\pi}_T, \lambda) \leqslant \min_\pi L(\pi, \widehat{\lambda}_T) + \omega$, which will terminate the algorithm.

Note that we always have $\max_\lambda L(\hat{\pi}_T, \lambda) \geqslant L(\hat{\pi}_T, \widehat{\lambda}_T) \geqslant \min_\pi L(\pi, \widehat{\lambda}_T)$. Algorithm 1's convergence rate depends on the regret bound of the `Online-algorithm` procedure. Multiple algorithms exist with regret scaling as $\Omega(\sqrt{T})$ (e.g., online gradient descent with regularizer, variants of online mirror descent). In that case, the algorithm will terminate after $O(\frac{1}{\omega^2})$ iterations.

Empirical Convergence Analysis of Main Algorithm - Proof of Theorem 2.4.1

By choosing normalized exponentiated gradient as the online learning subroutine, we have the following regret bound after T iterations of the main algorithm 2 (chapter 2 of (Shalev-Shwartz et al., 2012)) for any $\lambda \in \mathbb{R}_+^{m+1}$, $\|\lambda\|_1 = B$:

$$\frac{1}{T} \sum_{t=1}^{T} \widehat{L}(\pi_t, \lambda) \leqslant \frac{1}{T} \sum_{t=1}^{T} \widehat{L}(\pi_t, \lambda_t) + \frac{\frac{B \log(m+1)}{\eta} + \eta \bar{G}^2 BT}{T} \tag{A.5}$$

Denote $\omega_T = \frac{\frac{B\log(m+1)}{\eta} + \eta\bar{G}^2BT}{T}$ to simplify notations. By the linearity of $\widehat{L}(\pi,\lambda)$ in both π and λ, we have for any λ that

$$\widehat{L}(\hat{\pi}_T,\lambda) \overset{\text{linearity}}{=} \frac{1}{T}\sum_{t=1}^{T}\widehat{L}(\pi_t,\lambda) \overset{\text{eqn (A.5)}}{\leqslant} \frac{1}{T}\sum_{t=1}^{T}\widehat{L}(\pi_t,\lambda_t) + \omega_T$$
$$\overset{\text{best response } \pi_t}{\leqslant} \frac{1}{T}\sum_{t=1}^{T}\widehat{L}(\hat{\pi}_T,\lambda_t) + \omega_T \overset{\text{linearity}}{=} \widehat{L}(\hat{\pi}_T,\widehat{\lambda}_T) + \omega_T$$

Since this is true for any λ, $\max_\lambda \widehat{L}(\hat{\pi}_T,\lambda) \leqslant \widehat{L}(\hat{\pi}_T,\widehat{\lambda}_T) + \omega_T$.

On the other hand, for any policy π, we also have

$$\widehat{L}(\pi,\widehat{\lambda}_T) \overset{\text{linearity}}{=} \frac{1}{T}\sum_{t=1}^{T}\widehat{L}(\pi,\lambda_t) \overset{\text{best response } \pi_t}{\geqslant} \frac{1}{T}\sum_{t=1}^{T}\widehat{L}(\pi_t,\lambda_t)$$
$$\overset{\text{eqn (A.5)}}{\geqslant} \frac{1}{T}\sum_{t=1}^{T}\widehat{L}(\pi_t,\widehat{\lambda}_T) - \omega_T \overset{\text{linearity}}{=} \widehat{L}(\hat{\pi}_T,\widehat{\lambda}_T) - \omega_T$$

Thus $\min_\pi \widehat{L}(\pi,\widehat{\lambda}_T) \geqslant \widehat{L}(\hat{\pi}_T,\widehat{\lambda}_T) - \omega_T$, leading to

$$\max_\lambda \widehat{L}(\hat{\pi}_T,\lambda) - \min_\pi \widehat{L}(\pi,\widehat{\lambda}_T) \leqslant \widehat{L}(\hat{\pi}_T,\widehat{\lambda}_T) + \omega_T - (\widehat{L}(\hat{\pi}_T,\widehat{\lambda}_T) - \omega_T) = 2\omega_T$$

After T iterations of the main algorithm 2, therefore, the empirical primal-dual gap is bounded by

$$\max_\lambda \widehat{L}(\hat{\pi}_T,\lambda) - \min_\pi \widehat{L}(\pi,\widehat{\lambda}_T) \leqslant \frac{2\frac{B\log(m+1)}{\eta} + 2\eta\bar{G}^2BT}{T}$$

In particular, if we want the gap to fall below a desired threshold ω, setting the online learning rate $\eta = \frac{\omega}{4\bar{G}^2B}$ will ensure that the algorithm converges after $\frac{16B^2\bar{G}^2\log(m+1)}{\omega^2}$ iterations.

A.3 End-to-end Generalization Analysis of Main Algorithm

In this section, we prove the following full statement of theorem 2.4.4 of the main chapter 2. Note that to lessen notation, we define $\bar{V} = \bar{C} + B\bar{G}$ to be the bound of value functions under considerations in algorithm 2.

Theorem A.3.1 (Generalization bound of algorithm 2)**.** *Let π^* be the optimal policy to problem OPT. Let K be the number of iterations of FQE and FQI. Let $\hat{\pi}$ be the policy returned by our main algorithm 2, with termination threshold ω. For any $\epsilon > 0, \delta \in (0,1)$, when $n \geqslant \frac{24\cdot 214\cdot \bar{V}^4}{\epsilon^2}(\log\frac{K(m+1)}{\delta} + \mathtt{dim}_{\mathcal{F}}\log\frac{320\bar{V}^2}{\epsilon^2} + \log(14e(\mathtt{dim}_{\mathcal{F}}+1)))$, we have with probability at least $1-\delta$:*

$$C(\hat{\pi}) \leqslant C(\pi^*) + \omega + \frac{(4+B)\gamma}{(1-\gamma)^3}(\sqrt{C_\rho}\epsilon + 2\gamma^{K/2}\bar{V})$$

and

$$G(\hat{\pi}) \leqslant \tau + 2\frac{\bar{V}+\omega}{B} + \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}(\sqrt{C_\rho}\epsilon + \frac{2\gamma^{K/2}\bar{V}}{(1-\gamma)^{1/2}})$$

Let $\hat{\pi} = \frac{1}{T}\sum_t \pi_t$ be the returned policy T iterations, with corresponding dual variable $\widehat{\lambda} = \frac{1}{T}\sum_t \lambda_t$.

By the stopping condition, the empirical duality gap is less than some threshold ω, i.e., $\max_{\lambda\in\mathbb{R}_+^{m+1}, \|\lambda\|_1 = B} \widehat{L}(\hat{\pi},\lambda) - \min_{\pi\in\Pi}\widehat{L}(\pi,\widehat{\lambda}) \leqslant \omega$ where $\widehat{L}(\pi,\lambda) = \widehat{C}(\pi) + \lambda^\top(\widehat{G}(\pi) - \tau)$. We first show that the returned policy approximately satisfies the constraints. The proof of theorem A.3.1 will make use of the following empirical constraint satisfaction bound:

Lemma A.3.2 (Empirical constraint satisfactions)**.** *Assume that the constraints $\widehat{G}(\pi) \leqslant \tau$ are feasible. Then the returned policy $\hat{\pi}$ approximately satisfies all constraints*

$$\max_{i=1:m+1}(\widehat{g}_i(\hat{\pi}) - \tau_i) \leqslant 2\frac{\bar{C}+\omega}{B}$$

Proof. We consider $\max_{i=1:m+1}(\widehat{g}_i(\hat{\pi}) - \tau_i) > 0$ (otherwise the lemma statement is trivially true). The termination condition implies that $\widehat{L}(\hat{\pi},\widehat{\lambda}) - \max_{\lambda\in\mathbb{R}_+^{m+1}, \|\lambda\|_1 = B}\widehat{L}(\hat{\pi},\lambda) \geqslant -\omega$

$$\implies \widehat{\lambda}^\top(\widehat{G}(\hat{\pi}) - \widehat{\tau}) \geqslant \max_{\lambda\in\mathbb{R}_+^{m+1}, \|\lambda\|_1 = B}\lambda^\top(\widehat{G}(\hat{\pi}) - \widehat{\tau}) - \omega \tag{A.6}$$

Relaxing the RHS of equation (A.6) by setting $\lambda[j] = B$ for $j = \arg\max_{i=1:m+1} [\widehat{g}_i(\hat{\pi}) - \tau_i]$ and $\lambda[i] = 0 \ \ \forall i \neq j$ yields:

$$B \max_{i=1:m+1} [\widehat{g}_i(\hat{\pi}) - \tau_i] - \omega \leqslant \widehat{\lambda}^\top (\widehat{G}(\hat{\pi}) - \tau) \tag{A.7}$$

Given π such that $\widehat{G}(\pi) \leqslant \tau$, also by the termination condition:

$$\widehat{L}(\hat{\pi}, \widehat{\lambda}) - \widehat{L}(\pi, \widehat{\lambda}) \leqslant \max_{\lambda \in \mathbb{R}_+^{m+1}, \|\lambda\|_1 = B} \widehat{L}(\hat{\pi}, \lambda) - \min_{\pi \in \Pi} \widehat{L}(\pi, \widehat{\lambda}) \leqslant \omega$$

Thus implies

$$\widehat{L}(\hat{\pi}, \widehat{\lambda}) \leqslant \widehat{L}(\pi, \widehat{\lambda}) + \omega = \widehat{C}(\pi) + \widehat{\lambda}^\top (\widehat{G}(\pi) - \tau) \leqslant \widehat{C}(\pi) + \omega \tag{A.8}$$

combining what we have from equation (A.8) and (A.7):

$$B \max_{i=1:m+1} [\widehat{g}_i(\hat{\pi}) - \widehat{\tau}_i] - \omega \leqslant \widehat{\lambda}^\top (\widehat{G}(\hat{\pi}) - \widehat{\tau}) = \widehat{L}(\hat{\pi}, \widehat{\lambda}) - \widehat{C}(\hat{\pi}) \leqslant \widehat{C}(\pi) + \omega - \widehat{C}(\hat{\pi})$$

Rearranging and bounding $\widehat{C}(\pi) \leqslant \bar{C}$ and $\widehat{C}(\hat{\pi}) \leqslant -\bar{C}$ finishes the proof of the lemma. □

We now return to the proof of theorem A.3.1, our task is to lift empirical error to generalization bound for main objective and constraints.

Denote by ϵ_{FQE} the (generalization) error introduced by the Fitted Q Evaluation procedure (algorithm 3) and ϵ_{FQI} the (generalization) error introduced by the Fitted Q Iteration procedure (algorithm 15). For now we keep ϵ_{FQE} and ϵ_{FQI} unspecified (to be specified shortly). That is, for each $t = 1, 2, \ldots, T$, we have with probability at least $1 - \delta$:

$$C(\pi_t) + \lambda_t^\top (G(\pi_t) - \tau) \leqslant C(\pi^*) + \lambda_t^\top (G(\pi^*) - \tau) + \epsilon_{FQI}$$

Since π^* satisfies the constraints, i.e., $G(\pi^*) - \tau \leqslant 0$ componentwise, and $\lambda_t \geqslant 0$, we also have with probability $1 - \delta$

$$L(\pi_t, \lambda_t) = C(\pi_t) + \lambda_t^\top (G(\pi_t) - \tau) \leqslant C(\pi^*) + \epsilon_{FQI} \tag{A.9}$$

Similarly, with probability $1 - \delta$, all of the following inequalities are true

$$\widehat{C}(\pi_t) + \epsilon_{FQE} \geqslant C(\pi_t) \geqslant \widehat{C}(\pi_t) - \epsilon_{FQE} \tag{A.10}$$

$$\widehat{G}(\pi_t) + \epsilon_{FQE}\mathbf{1} \geqslant G(\pi_t) \geqslant \widehat{G}(\pi_t) - \epsilon_{FQE}\mathbf{1} \text{ (row wise for all } m \text{ constraints)} \tag{A.11}$$

Thus with probability at least $1-\delta$

$$\begin{aligned} L(\pi_t,\lambda_t) = C(\pi_t) + \lambda_t^\top (G(\pi_t)-\tau) &\geqslant \widehat{C}(\pi_t) + \lambda_t^\top (\widehat{G}(\pi_t)-\tau) - \epsilon_{FQE}(1+\lambda_t^\top \mathbf{1}) \\ &\geqslant \widehat{C}(\pi_t) + \lambda_t^\top (\widehat{G}(\pi_t)-\tau) - \epsilon_{FQE}(1+B) \\ &= \widehat{L}(\pi_t,\lambda_t) - \epsilon_{FQE}(1+B) \end{aligned} \tag{A.12}$$

Recall that the execution of mixture policy $\hat{\pi}$ is done by uniformly sampling one policy π_t from $\{\pi_1,\ldots,\pi_T\}$, and rolling-out with π_t. Thus from equations (A.9) and (A.12), we have $\mathbb{E}_{t\sim U[1:T]}\widehat{L}(\pi_t,\lambda_t) \leqslant C(\pi^*) + \epsilon_{FQI} + (1+B)\epsilon_{FQE}$ w.p. $1-\delta$. In other words, with probability $1-\delta$:

$$\frac{1}{T}\sum_{t=1}^{T} \widehat{L}(\pi_t,\lambda_t) \leqslant C(\pi^*) + \epsilon_{FQI} + (1+B)\epsilon_{FQE}$$

Due to the no-regret property of our online algorithm (EG in this case):

$$\frac{1}{T}\sum_{t=1}^{T} \widehat{L}(\pi_t,\lambda_t) \geqslant \max_{\lambda} \widehat{L}(\hat{\pi},\lambda) - \omega = \widehat{C}(\hat{\pi}) + \max_{\lambda} \lambda^\top (\widehat{G}(\hat{\pi})-\tau) - \omega$$

If $\widehat{G}(\hat{\pi})-\tau \leqslant 0$ componentwise, choose $\lambda[i]=0, i=1,2,\ldots,m$ and $\lambda[m+1]=B$. Otherwise, we can choose $\lambda[j]=B$ for $j=\arg\max_{i=1:m+1}[\widehat{g}_i(\hat{\pi})-\tau[i]]$ and $\lambda[i]=0\ \ \forall i\neq j$. We can see that $\max_{\lambda\in\mathbb{R}_+^{m+1},\|\lambda\|_1=B} \lambda^\top(\widehat{G}(\hat{\pi})-\tau) \geqslant 0$. Therefore:

$$\widehat{C}(\hat{\pi}) - \omega \leqslant C(\pi^*) + \epsilon_{FQI} + (1+B)\epsilon_{FQE} \text{ with probability at least } 1-\delta$$

Combined with the first term from equation (A.10):

$$C(\hat{\pi}) - \epsilon_{FQE} - \omega \leqslant C(\pi^*) + \epsilon_{FQI} + (1+B)\epsilon_{FQE}$$

or

$$C(\hat{\pi}) \leqslant C(\pi^*) + \omega + \epsilon_{FQI} + (2+B)\epsilon_{FQE} \tag{A.13}$$

We now bring in the generalization error results from our standalone analysis of FQI (appendix A.6) and FQE (appendix A.5) into equation (A.13).

Specifically, when
$n \geqslant \frac{24\cdot 214\cdot \bar{V}^4}{\epsilon^2}\left(\log\frac{K(m+1)}{\delta} + \texttt{dim}_{\mathcal{F}}\log\frac{320\bar{V}^2}{\epsilon^2} + \log(14e(\texttt{dim}_{\mathcal{F}}+1))\right)$, when FQI and FQE are run with K iterations, we have the guarantee that for any $\epsilon > 0$,

with probability at least $1-\delta$

$$C(\hat{\pi}) \leqslant C(\pi^*) + \omega + \underbrace{\frac{2\gamma}{(1-\gamma)^3}\left(\sqrt{C_\mu}\epsilon + 2\gamma^{K/2}\bar{V}\right)}_{\text{FQI generalization error}} + \\ + \underbrace{\frac{\gamma^{1/2}(2+B)}{(1-\gamma)^{3/2}}\left(\sqrt{C_\mu}\epsilon + \frac{\gamma^{K/2}}{(1-\gamma)^{1/2}}2\bar{V}\right)}_{(2+B)\times \text{ FQE generalization error}} \\ \leqslant C(\pi^*) + \omega + \frac{(4+B)\gamma}{(1-\gamma)^3}\left(\sqrt{C_\mu}\epsilon + 2\gamma^{K/2}\bar{V}\right) \tag{A.14}$$

From lemma A.3.2, $\widehat{G}(\hat{\pi}) \leqslant \tau + 2\frac{\bar{C}+\omega}{B} \leqslant \tau + 2\frac{\bar{V}+\omega}{B}$. From equation (A.11), for each t=1,2,. . . ,T, we have $\widehat{G}(\pi_t) \geqslant G(\pi_t) - \epsilon_{FQE}\mathbf{1}$ with probability $1-\delta$. Thus

$$\begin{aligned}\mathbf{P}\left(\widehat{G}(\hat{\pi}) \geqslant G(\hat{\pi}) - \epsilon_{FQE}\mathbf{1}\right) &= \sum_{t=1}^{T}\mathbf{P}(\widehat{G}(\pi_t) \\ &\geqslant G(\pi_t) - \epsilon_{FQE}\mathbf{1}|\hat{\pi}=\pi_t)\mathbf{P}(\hat{\pi}=\pi_t) \\ &\geqslant T(1-\delta)\frac{1}{T} = 1-\delta\end{aligned}$$

Therefore, we have the following generalization guarantee for the approximate satisfaction of all constraints:

$$G(\hat{\pi}) \leqslant \tau + 2\frac{\bar{V}+\omega}{B} + \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\left(\sqrt{C_\mu}\epsilon + \frac{\gamma^{K/2}}{(1-\gamma)^{1/2}}2\bar{V}\right) \tag{A.15}$$

Inequalities (A.14) and (A.15) complete the proof of theorem A.3.1 (and theorem 2.4.4 of the main thesis chapter 2)

A.4 Preliminaries to Analysis of Fitted Q Algorithms

In this section, we set-up necessary notations and definitions for the theoretical analysis of FQE and FQI. To simplify the presentation, we will focus exclusively on weighted ℓ_2 norm for error analysis.

With the definitions and assumptions presented in this section, we will present the sample complexity guarantee of Fitted-Q-Evaluation (FQE) in appendix A.5. The proof for FQI will follow similarly in appendix A.6.

While it is possible to adapt proofs from related algorithms (Munos and Szepesvári, 2008; Antos et al., 2008b) to analyze FQE and FQI, in the next two sections we show improved convergence rate from $O(n^{-4})$ to $O(n^{-2})$, where n is the number of samples in data set D.

To be consistent with the notations in the main thesis chapter 2, we use the convention $C(\pi)$ as the value function that denotes long-term accumulated cost, instead of using $V(\pi)$ denoting long-term rewards in the traditional RL literature. Our notation for Q function is similar to the RL literature - the only difference is that the optimal policy minimizes $Q(x,a)$ instead of maximizing. We denote the bound on the value function as $\bar{C}$ (alternatively if the single timestep cost is bounded by $\bar{c}$, then $\bar{C} = \frac{\bar{c}}{1-\gamma}$). For simplicity, the standalone analysis of FQE and FQI concerns only with the cost objective c. Dealing with cost $c + \lambda^\top g$ offers no extra difficulty - in that case we simply augment the bound of the value function to $\bar{V} = \bar{C} + B\bar{G}$.

Bellman operators

The *Bellman optimality operator* $\mathbb{T} : \mathcal{B}(\mathbf{X} \times \mathbf{A}; \bar{C}) \mapsto \mathcal{B}(\mathbf{X} \times \mathbf{A}; \bar{C})$ as

$$(\mathbb{T}Q)(x,a) = c(x,a) + \gamma \int_{\mathbf{X}} \min_{a' \in \mathbf{A}} Q(x',a') p(dx'|x,a) \tag{A.16}$$

The optimal value functions are defined as usual by $C^*(x) = \sup_\pi C^\pi(x)$ and $Q^*(x,a) = \sup_\pi Q^\pi(x,a) \;\; \forall x \in \mathbf{X}, a \in \mathbf{A}$.

For a given policy π, the *Bellman evaluation operator* $\mathbb{T}^\pi : \mathcal{B}(\mathbf{X} \times \mathbf{A}; \bar{C}) \mapsto \mathcal{B}(\mathbf{X} \times \mathbf{A}; \bar{C})$ as

$$(\mathbb{T}^\pi Q)(x,a) = c(x,a) + \gamma \int_{\mathbf{X}} Q(x', \pi(x')) p(dx'|x,a) \tag{A.17}$$

It is well known that $\mathbb{T}^\pi Q^\pi = Q^\pi$, a fixed point of the $\mathbb{T}^\pi$ operator.

Data distribution and weighted ℓ_2 norm

Denote the state-action data generating distribution as μ, induced by some data-generating (behavior) policy π_{D}, that is, $(x_i, a_i) \sim \mu$ for $(x_i, a_i, x_i', c_i) \in \mathrm{D}$.

Note that data set D is formed by multiple trajectories generated by π_{D}. For each (x_i, a_i), we have $x_i' \sim p(\cdot|x_i, a_i)$ and $c_i = c(x_i, a_i)$. For any (measurable) function $f : \mathbf{X} \times \mathbf{A} \mapsto \mathbb{R}$, define the μ-weighted ℓ_2 norm of f as $\|f\|_\mu^2 = \int_{\mathbf{X}\times\mathbf{A}} f(x,a)^2 \mu(dx, da) = \int_{\mathbf{X}\times\mathbf{A}} f(x,a)^2 \mu_x(dx)\pi_{\mathrm{D}}(a|dx)$. Similarly for any other state-action distribution ρ, $\|f\|_\rho^2 = \int_{\mathbf{X}\times\mathbf{A}} f(x,a)^2 \rho(dx, da)$

Inherent Bellman error

FQE and FQI depend on a chosen function class $\mathcal{F}$ to approximate $Q(x, a)$. To express how well the Bellman operator $\mathbb{T}g$ can be approximated by a function in the policy class $\mathcal{F}$, when $\mathbb{T}g \notin \mathcal{F}$, a notion of distance, known as inherent Bellman error was first proposed by (Munos, 2003) and used in the analysis of related ADP algorithms (Munos and Szepesvári, 2008; Munos, 2007; Antos et al., 2008a,b; Lazaric et al., 2010, 2012; Lazaric and Restelli, 2011; Maillard et al., 2010).

Definition A.4.1 (Inherent Bellman Error)**.** Given a function class $\mathcal{F}$ and a chosen distribution ρ, the *inherent Bellman error* of $\mathcal{F}$ is defined as

$$d_{\mathcal{F}} = d(\mathcal{F}, \mathbb{T}\mathcal{F}) = \sup_{h\in\mathcal{F}} \inf_{f\in\mathcal{F}} \|f - \mathbb{T}h\|_\rho$$

where $\|\cdot\|_\rho$ is the $\rho-$weighted ℓ_2 norm and $\mathbb{T}$ is the Bellman optimality operator defined in (A.16)

To analyze FQE, we will form a similar definition for the Bellman evaluation operator

Definition A.4.2 (Inherent Bellman Evaluation Error)**.** Given a function class $\mathcal{F}$ and a policy π, the *inherent Bellman evaluation error* of $\mathcal{F}$ is defined as

$$d_{\mathcal{F}}^\pi = d(\mathcal{F}, \mathbb{T}^\pi\mathcal{F}) = \sup_{h\in\mathcal{F}} \inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi h\|_{\rho_\pi}$$

where $\|\cdot\|_{\rho_\pi}$ is the ℓ_2 norm weighted by ρ_π. ρ_π is defined as the state-action distribution induced by policy π, and $\mathbb{T}^\pi$ is the Bellman operator defined in (A.17)

Concentrability coefficients

Let P^π denote the operator acting on $f : \mathbf{X} \times \mathbf{A} \mapsto \mathbb{R}$ such that $(P^\pi f)(x, a) = \int_{\mathbf{X}} f(x', \pi(x'))p(x'|x, a)dx'$. Acting on f (e.g., approximates Q), P^π captures the transition dynamics of taking action a and following π thereafters.

The following definition and assumption are standard in the analysis of related approximate dynamic programming algorithms (Lazaric et al., 2012; Munos and Szepesvári, 2008; Antos et al., 2008a). As approximate value iteration and policy iteration algorithms perform policy update, the new policy at each round will induce a different stationary state-action distribution. One way to quantify the distribution shift is the notion of concentrability coefficient of future state-action distribution, a variant of the notion introduced by (Munos, 2003).

Definition A.4.3 (Concentrability coefficient of state-action distribution)**.** Given data generating distribution $\mu \sim \pi_{\mathrm{D}}$, initial state distribution χ. For $m \geqslant 0$, and an arbitrary sequence of stationary policies $\{\pi_m\}_{m \geqslant 1}$ let

$$\beta_\mu(m) = \sup_{\pi_1,\ldots,\pi_m} \left\| \frac{d(\chi P^{\pi_1} P^{\pi_2} \ldots P^{\pi_m})}{d\mu} \right\|_\infty$$

($\beta_\mu(m) = \infty$ if the future state distribution $\chi P^{\pi_1} P^{\pi_2} \ldots P^{\pi_m}$ is not absolutely continuous w.r.t. μ, i.e, $\chi P^{\pi_1} P^{\pi_2} \ldots P^{\pi_m}(x,a) > 0$ for some $\mu(x,a) = 0$)

Assumption 3. $\beta_\mu = (1-\gamma)^2 \sum_{m \geqslant 1} m\gamma^{m-1}\beta_\mu(m) < \infty$

Combination Lock Example. An example of an MDP that violates Assumption 3 is the "combination lock" example proposed by (Koenig and Simmons, 1996). In this finite MDP, we have N states $\mathbf{X} = \{1, 2, \ldots, N\}$, and 2 actions: going L or R. The initial state is $x_0 = 1$. In any state x, action L takes agent back to initial state x_0, and action R advances the agent to the next state $x + 1$ in a chain fashion. Suppose that the reward is 0 everywhere except for the very last state N. One can see that for an MDP such that any behavior policy π_D that has a bounded from below probability of taking action L from any state x, i.e., $\pi_D(L|x) \geqslant \nu > 0$, then it takes an exponential number of trajectories to learn or evaluate a policy that always takes action R. In this setting, we can see that the concentration coefficient β_μ can be designed to be arbitrarily large.

Complexity measure of function class $\mathcal{F}$

Definition A.4.4 (Random L_1 Norm Covers)**.** Let $\epsilon > 0$, let $\mathcal{F}$ be a set of functions $\mathbf{X} \mapsto \mathbb{R}$, let $x_1^n = (x_1, \ldots, x_n)$ be n fixed points in $\mathbf{X}$. Then a collection of functions $\mathcal{F}_\epsilon = \{f_1, \ldots, f_N\}$ is an ϵ-cover of $\mathcal{F}$ on x_1^n if

$$\forall f \in \mathcal{F}, \exists f' \in \mathcal{F}_\epsilon : |\frac{1}{n}\sum_{i=1}^n f(x_i) - \frac{1}{n}\sum_{i=1}^n f'(x_i)| \leqslant \epsilon$$

The empirical covering number, denote by $\mathcal{N}_1(\epsilon, \mathcal{F}, x_1^n)$, is the size of the smallest ϵ-cover on x_1^n. Take $\mathcal{N}_1(\epsilon, \mathcal{F}, x_1^n) = \infty$ if no finite ϵ-cover exists.

Definition A.4.5 (Pseudo-Dimension)**.** A real-valued function class $\mathcal{F}$ has pseudo-dimension $\texttt{dim}_{\mathcal{F}}$ defined as the VC dimension of the function class induced by the sub-level set of functions of $\mathcal{F}$. In other words, define function class $\mathrm{H} = \{(x, y) \mapsto \mathrm{sign}(f(x) - y : f \in \mathcal{F}\}$, then

$$\texttt{dim}_{\mathcal{F}} = \texttt{VC-dimension}(\mathrm{H})$$

A.5 Analysis of Fitted Q Evaluation

In this section we prove the following statement for Fitted Q Evaluation (FQE).

Theorem A.5.1 (Guarantee for FQE - General Case (theorem 2.4.2 in chapter 2))**.** *Under Assumption 3, for $\epsilon > 0$ & $\delta \in (0,1)$, after K iterations of Fitted Q Evaluation (Algorithm 3), for $n = O(\frac{\bar{C}^4}{\epsilon^2}(\log \frac{K}{\delta} + \mathtt{dim}_{\mathcal{F}} \log \frac{\bar{C}^2}{\epsilon^2} + \log \mathtt{dim}_{\mathcal{F}}))$, we have with probability $1-\delta$:*

$$\left|C(\pi) - \widehat{C}(\pi)\right| \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}(\sqrt{\beta_\mu}\left(2d^{\pi}_{\mathcal{F}} + \epsilon\right) + \frac{2\gamma^{K/2}\bar{C}}{(1-\gamma)^{1/2}}).$$

Theorem A.5.2 (Guarantee for FQE - Bellman Realizable Case)**.** *Under Assumptions 3-4, for any $\epsilon > 0, \delta \in (0,1)$, after K iterations of Fitted Q Evaluation (Algorithm 3), when $n \geqslant \frac{24\cdot 214\cdot \bar{C}^4}{\epsilon^2}(\log \frac{K}{\delta} + \mathtt{dim}_{\mathcal{F}} \log \frac{320\bar{C}^2}{\epsilon^2} + \log(14e(\mathtt{dim}_{\mathcal{F}} + 1)))$, we have with probability $1-\delta$:*

$$\left|C(\pi) - \widehat{C}(\pi)\right| \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}(\sqrt{\beta_\mu}\epsilon + \frac{2\gamma^{K/2}\bar{C}}{(1-\gamma)^{1/2}})$$

We first focus on theorem A.5.2, analyzing FQE assuming a sufficiently rich function class $\mathcal{F}$ so that the Bellman evaluation update $\mathbb{T}^\pi$ is closed wrt $\mathcal{F}$ (thus inherent Bellman evaluation error is 0). We call this the *Bellman evaluation realizability assumption*. This assumption simplifies the presentation of our bounds and also simplifies the final error analysis of Algo. 2.

After analyzing FQE under this Bellman realizable setting, we will turn to error bound for general, non-realizable setting in theorem A.5.1 (also theorem 2.4.2 in the main body of the thesis chapter 2). The main difference in the non-realizable setting is the appearance of an extra term $d^{\pi}_{\mathcal{F}}$ our final bound.

Error bound for single iteration - Bellman realizable case

Assumption 4 (Bellman evaluation realizability)**.** *We consider function classes $\mathcal{F}$ sufficiently rich so that $\forall f, \mathbb{T}^\pi f \in \mathcal{F}$.*

We begin with the following result bounding the error for a single iteration of FQE, under "training" distribution $\mu \sim \pi_D$

Proposition A.5.3 (Error bound for single iteration)**.** *Let the functions in $\mathcal{F}$ also be bounded by $\bar{C}$, and let $\mathtt{dim}_{\mathcal{F}}$ denote the pseudo-dimension of the function class $\mathcal{F}$. We have with probability at least $1-\delta$:*

$$\left\|Q_k - \mathbb{T}^\pi Q_{k-1}\right\|_\mu < \epsilon$$

when $n \geqslant \frac{24\cdot 214\cdot \bar{C}^4}{\epsilon^2}\left(\log\frac{1}{\delta} + \dim_{\mathcal{F}} \log\frac{320\bar{C}^2}{\epsilon^2} + \log(14e(\dim_{\mathcal{F}}+1))\right)$

Remark. Note from proposition A.5.3 that the dependence of sample complexity n here on ϵ is $\widetilde{O}(\frac{1}{\epsilon^2})$, which is better than previously known analysis for Fitted Value Iteration (Munos and Szepesvári, 2008) and FittedPolicyQ (continuous version of Fitted Q Iteration (Antos et al., 2008a)) dependence of $\widetilde{O}(\frac{1}{\epsilon^4})$. The finite sample analysis of LSTD (Lazaric et al., 2010) showed an $\widetilde{O}(\frac{1}{\epsilon^2})$ dependence using linear function approximation. Here we prove similar convergence rate for general non-linear (bounded) function approximators.

Proof of Proposition A.5.3. Recall the training target in round k is $y_i = c_i + \gamma Q_{k-1}(x_i', \pi(x_i'))$ for $i = 1, 2, \ldots, n$, and $Q_k \in \mathcal{F}$ is the solution to the following regression problem:

$$Q_k = \arg\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}(f(x_i,a_i) - y_i)^2$$

Consider random variables $(x,a) \sim \mu$ and $y = c(x,a) + \gamma Q_{k-1}(x', \pi(x'))$ where $x' \sim p(\cdot|x,a)$. By this definition, $\mathbb{T}^\pi Q_{k-1}$ is the *regression function* that minimizes square loss $\min_{h:\mathbb{R}^{X\times A}\mapsto\mathbb{R}} \mathbb{E}|h(x,a) - y|^2$ out of all functions h (not necessarily in $\mathcal{F}$). This is due to $(\mathbb{T}^\pi Q_{k-1})(\tilde{x},\tilde{a}) = \mathbb{E}[y|x=\tilde{x}, a=\tilde{a}]$ by definition of the Bellman operator. Consider Q_{k-1} fixed and we now want to relate the learned function Q_k over finite set of n samples with the regression function over the whole data distribution via uniform deviation bound. We use the following lemma:

Lemma A.5.4 ((Györfi et al., 2006), theorem 11.4. Original version (Lee et al., 1996), theorem 3)**.** *Consider random vector* (X,Y) *and n i.i.d samples* (X_i,Y_i)*. Let* $m(x)$ *be the (optimal) regression function under square loss* $m(x) = \mathbb{E}[Y|X=x]$*. Assume* $|Y| \leqslant B$ *a.s. and* $B \leqslant 1$*. Let* $\mathcal{F}$ *be a set of function* $f : \mathbb{R}^d \mapsto \mathbb{R}$ *and let* $|f(x)| \leqslant B$*. Then for each* $n \geqslant 1$

$$\begin{aligned}
\mathbf{P}\Bigg\{ &\exists f \in \mathcal{F} : \mathbb{E}|f(X) - Y|^2 - \mathbb{E}|m(X) - Y|^2 \\
&- \frac{1}{n}\sum_{i=1}^{n}\left(|f(X_i) - Y_i|^2 - |m(X_i) - Y_i|^2\right) \geqslant \\
&\epsilon\cdot\left(\alpha + \beta + \mathbb{E}|f(X) - Y|^2 - \mathbb{E}|m(X) - Y|^2\right)\Bigg\} \\
&\leqslant 14 \sup_{x_1^n} \mathcal{N}_1\left(\frac{\beta\epsilon}{20B}, \mathcal{F}, x_1^n\right)\exp\left(-\frac{\epsilon^2(1-\epsilon)\alpha n}{214(1+\epsilon)B^4}\right)
\end{aligned}$$

where $\alpha, \beta > 0$ *and* $0 < \epsilon < 1/2$

To apply this lemma, first note that since $\mathbb{T}^\pi Q_{k-1}$ is the optimal regression function[1], we have

$$\begin{aligned}\mathbb{E}_\mu\left[(Q_k(x,a)-y)^2\right] &= \mathbb{E}_\mu\left[(Q_k(x,a)-\mathbb{T}^\pi Q_{k-1}(x,a)+\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right] \\ &= \mathbb{E}_\mu\left[(Q_k(x,a)-\mathbb{T}^\pi Q_{k-1}(x,a))^2\right] + \mathbb{E}_\mu\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right]\end{aligned}$$

thus

$$\begin{aligned}\|Q_k-\mathbb{T}^\pi Q_{k-1}\|_\mu^2 &= \mathbb{E}\left[(Q_k(x,a)-\mathbb{T}^\pi Q_{k-1}(x,a))^2\right] \\ &= \mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right]\end{aligned}$$

where by definition

$$\begin{aligned}\mathbb{E}\left[(Q_k(x,a)-\mathbb{T}^\pi Q_{k-1}(x,a))^2\right] &= \int (Q_k(x,a)-\mathbb{T}^\pi Q_{k-1}(x,a))^2\,\mu(dx,da) \\ &= \int (Q_k(x,a)-\mathbb{T}^\pi(x,a))^2 \mu_x(dx)\pi_{\mathrm{D}}(a|dx)\end{aligned}$$

[1] It is easy to see that if $m(x)=\mathbb{E}[y|x]$ is the regression function then for any function $f(x)$, we have $\mathbb{E}\left[(f(x)-m(x))(m(x)-y)=0\right]$

Next, given a fixed data set $\widetilde{D}_k \sim \mu$

$$\mathbf{P}\{\, \|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\mu^2 > \epsilon\} =$$

$$\mathbf{P}\Big\{\mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right] > \epsilon\Big\}$$

$$\leqslant \mathbf{P}\Big\{\mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right]$$
$$- 2\cdot\left(\frac{1}{n}\sum_{i=1}^n (Q_k(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}^\pi Q_{k-1}(x_i,a_i)-y_i)^2\right) > \epsilon\Big\} \tag{A.18}$$

$$= \mathbf{P}\Big\{\mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right]$$
$$- \frac{1}{n}\sum_{i=1}^n \left[(Q_k(x_i,a_i)-y_i)^2 - (\mathbb{T}^\pi Q_{k-1}(x_i,a_i)-y_i)^2\right]$$
$$> \frac{1}{2}(\epsilon + \mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right])\Big\} \tag{A.19}$$

$$\leqslant \mathbf{P}\Big\{\exists f \in \mathcal{F} : \mathbb{E}\left[(f(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right]$$
$$- \frac{1}{n}\sum_{i=1}^n \left[(f(x_i,a_i)-y_i)^2 - (\mathbb{T}^\pi Q_{k-1}(x_i,a_i)-y_i)^2\right]$$
$$\geqslant \frac{1}{2}(\frac{\epsilon}{2} + \frac{\epsilon}{2} + \mathbb{E}\left[(f(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right])\Big\}$$
$$\leqslant 14 \sup_{x_1^n} \mathcal{N}_1\left(\frac{\epsilon}{80\bar{C}}, \mathcal{F}, x_1^n\right)\cdot \exp\left(-\frac{n\epsilon}{24\cdot 214\bar{C}^4}\right) \tag{A.20}$$

Equation (A.18) uses the definition of $Q_k = \arg\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n (f(x_i,a_i)-y_i)^2$ and the fact that $\mathbb{T}^\pi Q_{k-1} \in \mathcal{F}$, thus making the extra term a positive addition. Equation (A.19) is due to rearranging the terms. Equation (A.20) is an application of lemma A.5.4. We can further bound the empirical covering number by invoking the following lemma due to Haussler (Haussler, 1995):

Lemma A.5.5 ((Haussler, 1995), Corollary 3)**.** *For any set X, any points $x^{1:n} \in \boldsymbol{X}^n$, any class $\mathcal{F}$ of functions on X taking values in $[0,\bar{C}]$ with pseudo-dimension $\mathtt{dim}_{\mathcal{F}} < \infty$, and any $\epsilon > 0$*

$$\mathcal{N}_1(\epsilon,\mathcal{F},x_1^n) \leqslant e(\mathtt{dim}_{\mathcal{F}}+1)\left(\frac{2e\bar{C}}{\epsilon}\right)^{\mathtt{dim}_{\mathcal{F}}}$$

Applying lemma A.5.5 to equation (A.20), we have the inequality

$$\mathbf{P}\left\{ \|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\mu^2 > \epsilon \right\} \leqslant 14 \cdot e \cdot (\mathtt{dim}_{\mathcal{F}}+1) \left(\frac{320\bar{C}^2}{\epsilon}\right)^{\mathtt{dim}_{\mathcal{F}}} \cdot \exp\left(-\frac{n\epsilon}{24 \cdot 214\bar{C}^4}\right) \tag{A.21}$$

We thus have that when
$n \geqslant \frac{24 \cdot 214 \cdot \bar{C}^4}{\epsilon^2}\left(\log\frac{1}{\delta} + \mathtt{dim}_{\mathcal{F}} \log\frac{320\bar{C}^2}{\epsilon^2} + \log(14e(\mathtt{dim}_{\mathcal{F}}+1))\right)$, then:
$\|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\rho < \epsilon$ with probability at least $1-\delta$. Notice that the dependence of sample complexity n here on ϵ is $\widetilde{O}(\frac{1}{\epsilon^2})$, which is better than previously known analyses for other approximate dynamic programming algorithms such as Fitted Value Iteration (Munos and Szepesvári, 2008), FittedPolicyQ (Antos et al., 2008b,a) with dependence of $O(\frac{1}{\epsilon^4})$.

Error bound for single iteration - Bellman non-realizable case

We now give similar error bound for the general case, where Assumption 4 does not hold. Consider the decomposition

$$\begin{aligned}
&\|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\mu^2 = \mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right] \\
&= \Bigg\{ \mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a)-y)^2\right] \\
&\qquad - 2 \cdot \left(\frac{1}{n}\sum_{i=1}^n (Q_k(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}^\pi Q_{k-1}(x_i,a_i)-y_i)^2\right) \Bigg\} \\
&\qquad + \left\{ 2 \cdot \left(\frac{1}{n}\sum_{i=1}^n (Q_k(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}^\pi Q_{k-1}(x_i,a_i)-y_i)^2\right) \right\} \\
&= \mathtt{component_1} + \mathtt{component_2}
\end{aligned}$$

Splitting the probability of error into two separate bounds. We saw from the previous section (equation (A.21)) that

$$\mathbf{P}(\mathtt{component\_1} > \epsilon/2) \leqslant 14 \cdot e \cdot (\mathtt{dim}_{\mathcal{F}}+1) \left(\frac{640\bar{C}^2}{\epsilon}\right)^{\mathtt{dim}_{\mathcal{F}}} \cdot \exp\left(-\frac{n\epsilon}{48 \cdot 214\bar{C}^4}\right) \tag{A.22}$$

We no longer have $\mathtt{component\_2} \leqslant 0$ since $\mathbb{T}^\pi Q_{k-1} \notin \mathcal{F}$.
Let $f^* = \arg\inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu^2$. Since
$Q_k = \arg\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n (f(x_i,a_i)-y_i)^2$, we can upper-bound $\mathtt{component\_2}$ by

$$\mathtt{component\_2} \leqslant 2 \cdot \left(\frac{1}{n}\sum_{i=1}^n (f^*(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}^\pi Q_{k-1}(x_i,a_i)-y_i)^2\right)$$

We can treat f^* as a fixed function, unlike random function Q_k, and use standard concentration inequalities to bound the empirical average from the expectation. Let random variable $z = ((x,a),y)$, $z_i = ((x_i,a_i),y_i), i = 1,\ldots,n$ and let

$$h(z) = (f^*(x,a) - y)^2 - (\mathbb{T}^\pi Q_{k-1}(x,a) - y)^2$$

We have $|h(z)| \leqslant 4\bar{C}^2$. We will derive a bound for

$$\mathbf{P}\left(\frac{1}{n}\sum_{i=1}^{n} h(z_i) - \mathbb{E}h(z) > \frac{\epsilon}{4} + \mathbb{E}h(z)\right)$$

using Bernstein inequality(Mohri et al., 2012). First, using the relationship $h(z) = (f^*(x,a) + \mathbb{T}^\pi Q_{k-1}(x,a) - 2y)(f^*(x,a) - \mathbb{T}^\pi Q_{k-1}(x,a))$, the variance of $h(z)$ can be bounded by a constant factor of $\mathbb{E}h(z)$, since

$$\begin{aligned}
\mathbf{Var}(h(z)) &\leqslant \mathbb{E}h(z)^2 \leqslant 16\bar{C}^2\mathbb{E}\left[(f^*(x,a) - \mathbb{T}^\pi Q_{k-1}(x,a))^2\right] \\
&= 16\bar{C}^2\left(\mathbb{E}\left[(f^*(x,a) - y)^2\right] - \mathbb{E}\left[(\mathbb{T}^\pi Q_{k-1}(x,a) - y)^2\right]\right) \qquad \text{(A.23)} \\
&= 16\bar{C}^2\mathbb{E}h(z) \qquad \text{(A.24)}
\end{aligned}$$

Equation (A.23) stems from $\mathbb{T}^\pi Q_{k-1}$ being the optimal regression function. Now we can apply equation (A.24) and Bernstein inequality to obtain

$$\begin{aligned}
&\mathbf{P}\left(\frac{1}{n}\sum_{i=1}^{n} h(z_i) - \mathbb{E}h(z) > \frac{\epsilon}{4} + \mathbb{E}h(z)\right) \\
&\leqslant \mathbf{P}\left(\frac{1}{n}\sum_{i=1}^{n} h(z_i) - \mathbb{E}h(z) > \frac{\epsilon}{4} + \frac{\mathbf{Var}(h(z))}{16\bar{C}^2}\right) \\
&\leqslant \exp\left(-\frac{n\left(\frac{\epsilon}{4} + \frac{\mathbf{Var}}{16\bar{C}^2}\right)^2}{2\mathbf{Var} + 2\frac{4\bar{C}^2}{3}\left(\frac{\epsilon}{4} + \frac{\mathbf{Var}}{16\bar{C}^2}\right)}\right) \\
&\leqslant \exp\left(-\frac{n\left(\frac{\epsilon}{4} + \frac{\mathbf{Var}}{16\bar{C}^2}\right)^2}{\left(32\bar{C}^2 + \frac{8\bar{C}^2}{3}\right)\left(\frac{\epsilon}{4} + \frac{\mathbf{Var}}{16\bar{C}^2}\right)}\right) = \exp\left(-\frac{n\left(\frac{\epsilon}{4} + \frac{\mathbf{Var}}{16\bar{C}^2}\right)}{32\bar{C}^2 + \frac{8\bar{C}^2}{3}}\right) \\
&\leqslant \exp\left(-\frac{1}{128 + \frac{32}{3}} \cdot \frac{n\epsilon}{\bar{C}^2}\right)
\end{aligned}$$

Thus

$$\mathbf{P}\left(2 \cdot \left[\frac{1}{n}\sum_{i=1}^{n} h(z_i) - 2\mathbb{E}h(z)\right] > \frac{\epsilon}{2}\right) \leqslant \exp\left(-\frac{3}{416} \cdot \frac{n\epsilon}{\bar{C}^2}\right) \qquad \text{(A.25)}$$

Now we have

$$\texttt{component\_2} \leqslant 2 \cdot \frac{1}{n}\sum_{i=1}^{n} h(z_i) = 2 \cdot \left[\frac{1}{n}\sum_{i=1}^{n} h(z_i) - 2\mathbb{E}h(z)\right] + 4\mathbb{E}h(z)$$

Using again the fact that $\mathbb{T}^\pi Q_{k-1}$ is the optimal regression function

$$\begin{aligned}
\mathbb{E}h(z) &= \mathbb{E}_D\left[(f^*(x,a) - y)^2\right] - \mathbb{E}_D\left[(\mathbb{T}^\pi Q_{k-1}(x,a) - y)^2\right] \\
&= \mathbb{E}_D\left[(f^*(x,a) - \mathbb{T}^\pi Q_{k-1}(x,a))^2\right] \\
&= \inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu^2 \qquad \text{(A.26)}
\end{aligned}$$

Combining equations (A.22), (A.25) and (A.26), we can conclude that

$$\begin{aligned}
&\mathbf{P}\Big\{ \|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\mu^2 - 4\inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu^2 > \epsilon \Big\} \\
&\leqslant 14 \cdot e \cdot (\texttt{dim}_\mathcal{F} + 1)\left(\frac{640\bar{C}^2}{\epsilon}\right)^{\texttt{dim}_\mathcal{F}} \cdot \exp\left(-\frac{n\epsilon}{48\cdot 214\bar{C}^4}\right) \\
&\quad + \exp\left(-\frac{3}{416}\cdot\frac{n\epsilon}{\bar{C}^2}\right)
\end{aligned}$$

thus implying

$$\begin{aligned}
&\mathbf{P}\Big\{ \|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\mu - 2\inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu > \epsilon \Big\} \\
&\leqslant 14 \cdot e \cdot (\texttt{dim}_\mathcal{F} + 1)\left(\frac{640\bar{C}^2}{\epsilon^2}\right)^{\texttt{dim}_\mathcal{F}} \cdot \exp\left(-\frac{n\epsilon^2}{48\cdot 214\bar{C}^4}\right) \\
&\quad + \exp\left(-\frac{3}{416}\cdot\frac{n\epsilon^2}{\bar{C}^2}\right) \qquad \text{(A.27)}
\end{aligned}$$

We now can further upper-bound the term
$2\inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu \leqslant 2\sup_{f'\in\mathcal{F}}\inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi f'\|_\mu = 2d_\mathcal{F}^\pi$ (the worst-case *inherent Bellman evaluation error*), leading to the final bound for the Bellman non-realizable case.

One may wish to further remove the inherent Bellman evaluation error from our error bound. However, counter-examples exist where the inherent Bellman error cannot generally be estimated using function approximation (see section 11.6 of (Sutton and Barto, 2018b)). Fortunately, inherent Bellman error can be driven to be small by choosing rich function class $\mathcal{F}$ (low bias), at the expense of more samples requirement (higher variance, through higher pseudo-dimension $\texttt{dim}_\mathcal{F}$).

While the bound in (A.27) looks more complicated than the Bellman realizable case in equation A.21, note that the convergence rate will still be $O(\frac{1}{n^2})$.

Bounding the error across iterations

Previous sub-sections A.5 and A.5 have analyzed the error of FQE for a single iteration in Bellman realizable and non-realizable case. We now analyze how errors from different iterations flow through the FQE algorithm. The proof borrows the idea from lemma 3 and 4 of (Munos and Szepesvári, 2008) for fitted value iteration (for value function V instead of Q), with appropriate modifications for our off-policy evaluation context.

Recall that C^π, Q^π denote the true value function and action-value function, respectively, under the evaluation policy π. And $C_K = \mathbb{E}[Q_K(x, \pi(x))]$ denote the value function associated with the returned function Q_K from algorithm 3. Our goal is to bound the difference $C^\pi - C_K$ between the true value function and the estimated value of the returned function Q_K.

Let the unknown state-action distribution induced by the evaluation policy π be ρ. We first bound the loss $\|Q^\pi - Q_K\|_\rho$ under the "test-time "distribution ρ of (x, a), which differs from the state-action μ induced by data-generating policy π_{D}. We will then lift the loss bound from Q_K to C_K.

Step 1: Upper-bound the value estimation error

Let $\epsilon_{k-1} = Q_k - \mathbb{T}^\pi Q_{k-1} \in \mathbf{X} \times \mathbf{A}, \bar{C}$. We have for every k that

$$\begin{aligned} Q^\pi - Q_k &= \mathbb{T}^\pi Q^\pi - \mathbb{T}^\pi Q_{k-1} + \epsilon_{k-1} \quad (Q^\pi \text{ is fixed point of } T^\pi) \\ &= \gamma P^\pi (Q^\pi - Q_{k-1}) + \epsilon_{k-1} \end{aligned}$$

Thus by simple recursion

$$\begin{aligned} Q^\pi - Q_K &= \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^\pi)^{K-k-1} \epsilon_k + \gamma^K (P^\pi)^K (Q^\pi - Q_0) \\ &= \frac{1-\gamma^{K+1}}{1-\gamma} \Big[\sum_{k=0}^{K-1} \frac{(1-\gamma)\gamma^{K-k-1}}{1-\gamma^{K+1}} (P^\pi)^{K-k-1} \epsilon_k + \\ &\qquad \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}} (P^\pi)^K (Q^\pi - Q_0) \Big] \\ &= \frac{1-\gamma^{K+1}}{1-\gamma} \left[\sum_{k=0}^{K-1} \alpha_k A_k \epsilon_k + \alpha_K A_K (Q^\pi - Q_0) \right] \end{aligned} \tag{A.28}$$

where for simplicity of notations, we denote

$$\begin{aligned} \alpha_k &= \frac{(1-\gamma)\gamma^{K-k-1}}{1-\gamma^{K+1}} \text{ for } k < K, \alpha_K = \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}} \\ A_k &= (P^\pi)^{K-k-1}, A_K = (P^\pi)^K \end{aligned}$$

Note that A_k's are probability kernels and α_k's are deliberately chosen such that $\sum_k \alpha_k = 1$.

We can apply point-wise absolute value on both sides of (A.28) with $|f|$ being the short-hand notation for $|f(x,a)|$ and inequality holds point-wise. By triangle inequalities:

$$|Q^\pi - Q_K| \leqslant \frac{1-\gamma^{K+1}}{1-\gamma}\left[\sum_{k=0}^{K-1}\alpha_k A_k|\epsilon_k| + \alpha_K A_K|Q^\pi - Q_0|\right] \tag{A.29}$$

Step 2: Bounding $\|Q^\pi - Q_K\|_\rho$ for any unknown distribution ρ. To handle distribution shift from μ to ρ, we decompose the loss as follows:

$$\begin{aligned}
&\|Q^\pi - Q_K\|_\rho^2 = \int \rho(dx,da)\left(Q^\pi(x,a) - Q_K(x,a)\right)^2 \\
&\leqslant \left[\frac{1-\gamma^{K+1}}{1-\gamma}\right]^2 \int \rho(dx,da)\left[\left(\sum_{k=0}^{K-1}\alpha_k A_k|\epsilon_k| + \alpha_K A_K|Q^\pi - Q_0|\right)(x,a)\right]^2 \\
&\quad \text{(from}(A.29)\text{)} \\
&\leqslant \left[\frac{1-\gamma^{K+1}}{1-\gamma}\right]^2 \int \rho(dx,da)\left[\sum_{k=0}^{K-1}\alpha_k (A_k\epsilon_k)^2 + \alpha_K (A_K(Q^* - Q_0))^2\right](x,a) \\
&\quad \text{(Jensen)} \\
&\leqslant \left[\frac{1-\gamma^{K+1}}{1-\gamma}\right]^2 \int \rho(dx,da)\left[\sum_{k=0}^{K-1}\alpha_k A_k\epsilon_k^2 + \alpha_K A_K (Q^* - Q_0)^2\right](x,a) \text{ (Jensen)}
\end{aligned}$$

Using assumption 3 (assumption 1 of the main thesis chapter 2), we can bound each term ρA_k as

$$\rho A_k = \rho (P^\pi)^{K-k-1} \leqslant \mu\beta_\mu(K-k-1) \text{ (definition A.4.3)}$$

Thus

$$\begin{aligned}
&\|Q^\pi - Q_K\|_\rho^2 \leqslant \\
&\left[\frac{1-\gamma^{K+1}}{1-\gamma}\right]^2 \left[\frac{1}{1-\gamma^{K+1}}\sum_{k=0}^{K-1}(1-\gamma)\gamma^{K-k-1}\beta_\mu(K-k-1)\|\epsilon_k\|_\mu^2 + \alpha_K(2\bar{C})^2\right]
\end{aligned}$$

Assumption 3 (stronger than necessary for proof of FQE) can be used to upper-bound the first order concentration coefficient:

$$(1-\gamma)\sum_{m\geqslant 0}\gamma^m\beta_\mu(m) \leqslant \frac{\gamma}{1-\gamma}\left[(1-\gamma)^2\sum_{m\geqslant 1} m\gamma^{m-1}\beta_\mu(m)\right] = \frac{\gamma}{1-\gamma}\beta_\mu$$

This gives the upper-bound for $\|Q^\pi - Q_K\|_\rho^2$ as

$$\begin{aligned}
&\|Q^\pi - Q_K\|_\rho^2 \leqslant \\
&\left[\frac{1-\gamma^{K+1}}{1-\gamma}\right]^2 \left[\frac{\gamma}{(1-\gamma)(1-\gamma^{K+1})}\beta_\mu \max_k \|\epsilon_k\|_\mu^2 + \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}}(2\bar{C})^2\right] \\
&\leqslant \frac{1-\gamma^{K+1}}{(1-\gamma)^2}\left[\frac{\gamma}{1-\gamma}\beta_\mu \max_k \|\epsilon_k\|_\mu^2 + (1-\gamma)\gamma^K(2\bar{C})^2\right] \\
&\leqslant \frac{\gamma}{(1-\gamma)^3}\beta_\mu \max_k \|\epsilon_k\|_\mu^2 + \frac{\gamma^K}{1-\gamma}(2\bar{C})^2
\end{aligned}$$

Using $a^2 + b^2 \leqslant (a+b)^2$ for nonnegative a, b, we conclude that

$$\|Q^\pi - Q_K\|_\rho \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\left(\sqrt{\beta_\mu}\max_k \|\epsilon_k\|_\mu + \frac{\gamma^{K/2}}{(1-\gamma)^{1/2}}2\bar{C}\right) \tag{A.30}$$

Step 3: Turning error bound from Q to $|C^\pi - C_K|$ Now we can choose ρ to be the state-action distribution by the evaluation policy π. The error bound on the value function C follows simply by integrating inequality (A.30) over state-action pairs induced by π. The final error across iterations can be related to individual iteration error by

$$|C^\pi - C_K| \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\left(\sqrt{\beta_\mu}\max_k \|\epsilon_k\|_\mu + \frac{\gamma^{K/2}}{(1-\gamma)^{1/2}}2\bar{C}\right) \tag{A.31}$$

Finite-sample guarantees for Fitted Q Evaluation

Combining results from (A.21), (A.27) and (A.31), we have the final guarantees for FQE under both realizable and general cases.

Realizable Case - Proof of theorem A.5.2. From (A.21), when $n \geqslant \frac{24\cdot 214\cdot \bar{C}^4}{\epsilon^2}\left(\log\frac{K}{\delta} + \mathtt{dim}_{\mathcal{F}}\log\frac{320\bar{C}^2}{\epsilon^2} + \log(14e(\mathtt{dim}_{\mathcal{F}}+1))\right)$, we have $\|\epsilon_k\|_\mu < \epsilon$ with probability at least $1 - \delta/K$ for any $0 \leqslant k < K$. Thus we conclude that for any $\epsilon > 0, 0 < \delta < 1$, after K iterations of Fitted Q Evaluation, the value estimate returned by Q_K satisfies:

$$|C^\pi - C_K| \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\left(\sqrt{\beta_\mu}\epsilon + \frac{\gamma^{K/2}}{(1-\gamma)^{1/2}}2\bar{C}\right)$$

holds with probability $1-\delta$ when $n \geqslant \frac{24\cdot 214\cdot \bar{C}^4}{\epsilon^2}\left(\log\frac{K}{\delta} + \mathtt{dim}_{\mathcal{F}}\log\frac{320\bar{C}^2}{\epsilon^2} + \log(14e(\mathtt{dim}_{\mathcal{F}}+1))\right)$. This concludes the proof of theorem A.5.2.

Non-realizable Case - Proof of theorem A.5.1 and theorem 2.4.2 of chapter 2. Similarly, from (A.27) we have

$$\begin{aligned}
&\mathbf{P}\Big\{ \|Q_k - \mathbb{T}^\pi Q_{k-1}\|_\mu - 2 \inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu > \epsilon \Big\} \qquad \text{(A.32)}\\
&\leqslant 14 \cdot e \cdot (\dim_{\mathcal{F}} + 1) \left(\frac{640\bar{C}^2}{\epsilon^2}\right)^{\dim_{\mathcal{F}}} \cdot \exp\left(-\frac{n\epsilon^2}{48\cdot 214\bar{C}^4}\right)\\
&\quad + \exp\left(-\frac{3}{416}\cdot\frac{n\epsilon^2}{\bar{C}^2}\right)
\end{aligned}$$

Since $\inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi Q_{k-1}\|_\mu \leqslant \sup_{h\in\mathcal{F}} \inf_{f\in\mathcal{F}} \|f - \mathbb{T}^\pi h\|_\mu = d^\pi_{\mathcal{F}}$ (the *inherent Bellman evaluation error*), similar arguments to the realizable case lead to the conclusion that for any $\epsilon > 0, 0 < \delta < 1$, after K iterations of FQE:

$$|C^\pi - C_K| \leqslant \frac{\gamma^{1/2}}{(1-\gamma)^{3/2}}\left(\sqrt{\beta_\mu}(2d^\pi_{\mathcal{F}} + \epsilon) + \frac{\gamma^{K/2}}{(1-\gamma)^{1/2}} 2\bar{C}\right)$$

holds with probability $1-\delta$ when $n = O\big(\frac{\bar{C}^4}{\epsilon^2}(\log\frac{K}{\delta} + \dim_{\mathcal{F}} \log\frac{\bar{C}^2}{\epsilon^2} + \log\dim_{\mathcal{F}})\big)$, thus finishes the proof of theorem A.5.1.

Note that in both cases, the $\widetilde{O}(\frac{1}{\epsilon^2})$ dependency of n is significant improvement over previous finite-sample analysis of related approximate dynamic programming algorithms (Munos and Szepesvári, 2008; Antos et al., 2008b,a). This dependency matches that of previous analysis using linear function approximators from (Lazaric et al., 2012, 2010) for LSTD and LSPI algorithms. Here our analysis, using similar assumptions, is applicable for general non-linear, bounded function classes. , which is an improvement over convergence rate of $O(\frac{1}{n^4})$ in related approximate dynamic programming algorithms (Antos et al., 2008a,b; Munos and Szepesvári, 2008).

A.6 Analysis of Fitted Q Iteration

Algorithm and Discussion

Algorithm 15 Fitted Q Iteration with Function Approximation: $\texttt{FQI}(c)$ (Ernst et al., 2005)

Input: Collected data set $\mathrm{D} = \{x_i, a_i, x_i', c_i\}_{i=1}^n$. Function class $\mathcal{F}$

1: Initialize $Q_0 \in \mathcal{F}$ randomly
2: **for** $k = 1, 2, \ldots, K$ **do**
3: Compute target $y_i = c_i + \gamma \min_a Q_{k-1}(x_i', a) \;\; \forall i$
4: Build training set $\widetilde{\mathrm{D}}_k = \{(x_i, a_i), y_i\}_{i=1}^n$
5: Solve a supervised learning problem:
$Q_k = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n (f(x_i, a_i) - y_i)^2$
6: **end for**

Output: $\pi_K(\cdot) = \arg\min_a Q_K(\cdot, a)$ (greedy policy with respect to the returned function Q_K)

The analysis of FQI (algorithm 15) follows analogously from the analysis of FQE from the previous section (Appendix A.5). For brevity, we skip certain detailed derivations, especially those that are largely identical to FQE's analysis.

To the best of our knowledge, a finite-sample analysis of FQI with general non-linear function approximation has not been published (Continuous FQI from (Antos et al., 2008a) is in fact a Fitted Policy Iteration algorithm and is different from algo 15). In principle, one can adapt existing analysis of fitted value iteration (Munos and Szepesvári, 2008) and FittedPolicyQ (Antos et al., 2008b,a) to show that under similar assumptions, among policies greedy w.r.t. functions in $\mathcal{F}$, FQI will find $\epsilon-$ optimal policy using $n = \widetilde{O}(\frac{1}{\epsilon^4})$ samples. We derive an improved analysis of FQI with general non-linear function approximations, with better sample complexity of $n = \widetilde{O}(\frac{1}{\epsilon^2})$. We note that the appendix of (Lazaric and Restelli, 2011) contains an analysis of LinearFQI showing similar rate to ours, albeit with linear function approximators.

In this section, we prove the following statement:

Theorem A.6.1 (Guarantee for FQI - General Case (theorem 2.4.3 in chapter 2))**.** *Under Assumption 3, for any $\epsilon > 0, \delta \in (0,1)$, after K iterations of Fitted Q Iteration (algorithm 15), for $n = O\big(\frac{\bar{C}^4}{\epsilon^2}(\log \frac{K}{\delta} + \texttt{dim}_{\mathcal{F}} \log \frac{\bar{C}^2}{\epsilon^2} + \log \texttt{dim}_{\mathcal{F}})\big)$, we have with probability $1-\delta$:*

$$C^* - C(\pi_K) \leqslant \frac{2\gamma}{(1-\gamma)^3}\left(\sqrt{\beta_\mu}\,(2d_{\mathcal{F}} + \epsilon) + 2\gamma^{K/2}\bar{C}\right)$$

where π_K is the policy greedy with respect to the returned function Q_K, and C^* is the value of the optimal policy.

The key steps to the proof follow similar scheme to the proof of FQE. We first bound the error for each iteration, and then analyze how the errors flow through the algorithm.

Single iteration error bound $\|Q_k - \mathbb{T}Q_{k-1}\|_\mu$

Here μ is the state-action distribution induced by the data-generating policy π_{D}.

We begin with the decomposition:

$$\begin{aligned}
&\|Q_k - \mathbb{T}Q_{k-1}\|_\mu^2 = \mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}Q_{k-1}(x,a)-y)^2\right] \\
&= \Bigg\{\mathbb{E}\left[(Q_k(x,a)-y)^2\right] - \mathbb{E}\left[(\mathbb{T}Q_{k-1}(x,a)-y)^2\right] - \\
&2\cdot\left(\frac{1}{n}\sum_{i=1}^n (Q_k(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}Q_{k-1}(x_i,a_i)-y_i)^2\right)\Bigg\} \\
&\qquad + \left\{2\cdot\left(\frac{1}{n}\sum_{i=1}^n (Q_k(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}Q_{k-1}(x_i,a_i)-y_i)^2\right)\right\} \\
&= \texttt{component_1} + \texttt{component_2}
\end{aligned}$$

For $\mathbb{T}$ the Bellman (optimality) operator (equation A.16), $\mathbb{T}Q_{k-1}$ is the *regression function* that minimizes square loss $\min_{h:\mathbb{R}^{X\times A}\mapsto\mathbb{R}} \mathbb{E}|h(x,a)-y|^2$, with the random variables $(x,a)\sim\mu$ and $y = c(x,a) + \gamma \min_{a'} Q_{k-1}(x',a')$ where $x'\sim p(x'|x,a)$. Invoking lemma A.5.4 and following the steps similar to equations (A.18),(A.19),(A.20) and (A.21) from appendix A.5, we can bound the first component as

$$\mathbf{P}(\texttt{component\_1} > \epsilon/2) \leqslant 14\cdot e\cdot(\dim_{\mathcal{F}}+1)\left(\frac{640\bar{C}^2}{\epsilon}\right)^{\dim_{\mathcal{F}}}\cdot\exp\left(-\frac{n\epsilon}{48\cdot 214\bar{C}^4}\right) \tag{A.33}$$

Let $f^* = \inf_{f\in\mathcal{F}} \|f - \mathbb{T}Q_{k-1}\|_\mu^2$. Since $Q_k = \min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n (f(x_i,a_i)-y_i)^2$, we can upper-bound component_2 by

$$\texttt{component\_2} \leqslant 2\cdot\left(\frac{1}{n}\sum_{i=1}^n (f^*(x_i,a_i)-y_i)^2 - \frac{1}{n}\sum_{i=1}^n (\mathbb{T}Q_{k-1}(x_i,a_i)-y_i)^2\right)$$

Let random variable $z=((x,a),y)$, $z_i=((x_i,a_i),y_i), i=1,\ldots,n$ and let

$$h(z) = (f^*(x,a)-y)^2 - (\mathbb{T}Q_{k-1}(x,a)-y)^2$$

We have $|h(z)| \leqslant 4\bar{C}^2$. We can derive a bound for
$\mathbf{P}\left(\frac{1}{n}\sum_{i=1}^{n} h(z_i) - \mathbb{E}h(z) > \frac{\epsilon}{4} + \mathbb{E}h(z)\right)$ using Bernstein inequality, similar to equations (A.23) and (A.24) from appendix A.5 to obtain:

$$\mathbf{P}\left(2 \cdot \left[\frac{1}{n}\sum_{i=1}^{n} h(z_i) - 2\mathbb{E}h(z)\right] > \frac{\epsilon}{2}\right) \leqslant \exp\left(-\frac{3}{416} \cdot \frac{n\epsilon}{\bar{C}^2}\right) \tag{A.34}$$

Now we have

$$\texttt{component\_2} \leqslant 2 \cdot \frac{1}{n}\sum_{i=1}^{n} h(z_i) = 2 \cdot \left[\frac{1}{n}\sum_{i=1}^{n} h(z_i) - 2\mathbb{E}h(z)\right] + 4\mathbb{E}h(z)$$

Since

$$\begin{aligned}
\mathbb{E}h(z) &= \mathbb{E}_{\widetilde{D}_k}\left[(f^*(x,a) - y)^2\right] - \mathbb{E}_{\widetilde{D}_k}\left[(\mathbb{T}Q_{k-1}(x,a) - y)^2\right] \\
&= \mathbb{E}_{\widetilde{D}_k}\left[(f^*(x,a) - \mathbb{T}Q_{k-1}(x,a))^2\right] \\
&= \inf_{f \in \mathcal{F}} \|f - \mathbb{T}Q_{k-1}\|_\mu^2
\end{aligned} \tag{A.35}$$

Combining equations (A.33), (A.34) and (A.35), we obtain that

$$\begin{aligned}
&\mathbf{P}\Big\{\|Q_k - \mathbb{T}Q_{k-1}\|_\mu^2 - 4\inf_{f \in \mathcal{F}} \|f - \mathbb{T}Q_{k-1}\|_\mu^2 > \epsilon\Big\} \leqslant \\
&14 \cdot e \cdot (\texttt{dim}_{\mathcal{F}} + 1)\left(\frac{640\bar{C}^2}{\epsilon}\right)^{\texttt{dim}_{\mathcal{F}}} \cdot \exp\left(-\frac{n\epsilon}{48 \cdot 214\bar{C}^4}\right) \\
&\qquad + \exp\left(-\frac{3}{416} \cdot \frac{n\epsilon}{\bar{C}^2}\right)
\end{aligned} \tag{A.36}$$

Propagation of error bound for $\|Q^* - Q^{\pi_K}\|_\rho$

The analysis of error propagation for FQI is more involved than that of FQE, but the proof largely follows the error propagation analysis in lemma 3 and 4 of (Munos and Szepesvári, 2008) in the fitted value iteration context (for V function). We include the Q function's (slighly more complicated) derivation here for completeness.

Recall that π_K is greedy wrt the learned function Q_K returned by FQI. We aim to bound the difference $C^* - C^{\pi_K}$ between the optimal value function and that π_K. For a (to-be-specified) distribution ρ of state-action pairs (different from the data distribution μ), we bound the generalization loss $\|Q^* - Q^{\pi_K}\|_\rho$

Step 1: Upper-bound the propagation error (value). Let $\epsilon_{k-1} = Q_k - \mathbb{T}Q_{k-1}$.

We have that

$$\begin{aligned}
Q^* - Q_k &= \mathbb{T}^{\pi^*} Q^* - \mathbb{T}^{\pi^*} Q_{k-1} + \mathbb{T}^{\pi^*} Q_{k-1} - \mathbb{T} Q_{k-1} + \epsilon_{k-1} \\
&\leqslant \mathbb{T}^{\pi^*} Q^* - \mathbb{T}^{\pi^*} Q_{k-1} + \epsilon_{k-1} \text{ (b/c } \mathbb{T} Q_{k-1} \geqslant \mathbb{T}^{\pi^*} Q_{k-1}) \\
&= \gamma P^{\pi^*} (Q^* - Q_{k-1}) + \epsilon_{k-1}
\end{aligned}$$

Thus by recursion $Q^* - Q_K \leqslant \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^{\pi^*})^{K-k-1} \epsilon_k + \gamma^K (P^{\pi^*})^K (Q^* - Q_0)$

Step 2: Lower-bound the propagation error (value). Similarly

$$\begin{aligned}
Q^* - Q_k &= \mathbb{T} Q^* - \mathbb{T}^{\pi_{k-1}} Q^* + \mathbb{T}^{\pi_{k-1}} Q^* - \mathbb{T} Q_{k-1} + \epsilon_{k-1} \\
&\geqslant \mathbb{T}^{\pi_{k-1}} Q^* - \mathbb{T} Q_{k-1} + \epsilon_{k-1} \text{ (as } \mathbb{T} Q^* \geqslant \mathbb{T}^{\pi_{k-1}} Q^*) \\
&\geqslant \mathbb{T}^{\pi_{k-1}} Q^* - \mathbb{T}^{\pi_{k-1}} Q_{k-1} + \epsilon_{k-1} \text{ (b/c } \pi_{k-1} \text{ greedy wrt } Q_{k-1}) \\
&= \gamma P^{\pi_{k-1}} (Q^* - Q_{k-1}) + \epsilon_{k-1}
\end{aligned}$$

And by recursion

$$\begin{aligned}
Q^* - Q_K \geqslant \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^{\pi_{K-1}} P^{\pi_{K-2}} \dots P^{\pi_{k+1}}) \epsilon_k + \\
\gamma^K (P^{\pi_{K-1}} P^{\pi_{K-2}} \dots P^{\pi_0}) (Q^* - Q_0)
\end{aligned}$$

Step 3: Upper-bound the propagation error (policy). Beginning with a decomposition of value wrt to policy π_K

$$\begin{aligned}
Q^* - Q^{\pi_K} &= \mathbb{T}^{\pi^*} Q^* - \mathbb{T}^{\pi^*} Q_K + \mathbb{T}^{\pi^*} Q_K - \mathbb{T}^{\pi_K} Q_K + \mathbb{T}^{\pi_K} Q_K - \mathbb{T}^{\pi_K} Q^{\pi_K} \\
&\leqslant (\mathbb{T}^{\pi^*} Q^* - \mathbb{T}^{\pi^*} Q_K) + (\mathbb{T}^{\pi_K} Q_K - \mathbb{T}^{\pi_K} Q^{\pi_K}) \\
&\qquad (\text{ since } \mathbb{T}^{\pi^*} Q_K \leqslant \mathbb{T} Q_K = \mathbb{T}^{\pi_K} Q_K) \\
&= \gamma P^{\pi^*} (Q^* - Q_K) + \gamma P^{\pi_K} (Q_K - Q^{\pi_K}) \\
&= \gamma P^{\pi^*} (Q^* - Q_K) + \gamma P^{\pi_K} (Q_K - Q^* + Q^* - Q^{\pi_K})
\end{aligned}$$

Thus leading to $(I - \gamma P^{\pi_K})(Q^* - Q^{\pi_K}) \leqslant \gamma (P^{\pi^*} - P^{\pi_K})(Q^* - Q_K)$ The operator $(I - \gamma P^{\pi_K})$ is invertible and $(I - \gamma P^{\pi_K})^{-1} = \sum_{m \geqslant 0} \gamma^m (P^{\pi_K})^m$ is monotonic. Thus

$$\begin{aligned}
Q^* - Q^{\pi_K} &\leqslant \gamma (I - \gamma P^{\pi_K})^{-1} (P^{\pi^*} - P^{\pi_K})(Q^* - Q_K) \\
&= \gamma (I - \gamma P^{\pi_K})^{-1} P^{\pi^*} (Q^* - Q_K) - \gamma (I - \gamma P^{\pi_K})^{-1} P^{\pi_K} (Q^* - Q_K)
\end{aligned} \tag{A.37}$$

Applying inequalities from Step 1 and Step 2 to the RHS of (A.37), we have

$$Q^* - Q^{\pi_K} \leqslant (I - \gamma P^{\pi_K})^{-1} \Bigg[\sum_{k=0}^{K-1} \gamma^{K-k} \left((P^{\pi^*})^{K-k} - P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_{k+1}} \right) \epsilon_k$$
$$+ \gamma^{K+1} \left((P^{\pi^*})^{K+1} - (P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_0}) \right) (Q^* - Q_0) \Bigg] \tag{A.38}$$

Next we apply point-wise absolute value on RHS of (A.38), with $|\epsilon_k|$ being the shorthand notation for $|\epsilon_k(x,a)|$ point-wise. Using triangle inequalities and rewriting (A.38) in a more compact form ((Munos and Szepesvári, 2008)):

$$Q^* - Q^{\pi_K} \leqslant \frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2} \left[\sum_{k=0}^{K-1} \alpha_k A_k |\epsilon_k| + \alpha_K A_K |Q^* - Q_0| \right]$$

where $\alpha_k = \frac{(1-\gamma)\gamma^{K-k-1}}{1-\gamma^{K+1}}$ for $k < K, \alpha_K = \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}}$ and

$$A_k = \frac{1-\gamma}{2} (I - \gamma P^{\pi_K})^{-1} \left[(P^{\pi^*})^{K-k} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_{k+1}} \right] \text{ for } k < K$$
$$A_K = \frac{1-\gamma}{2} (I - \gamma P^{\pi_K})^{-1} \left[(P^{\pi^*})^{K+1} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_0} \right]$$

Note that A_k's are probability kernels that combine the P^{π_i} terms and α_k's are chosen such that $\sum_k \alpha_k = 1$.

Step 4: Bounding $\|Q^* - Q^{\pi_K}\|_\rho^2$ for any test distribution ρ.

This step handles distribution shift from μ to ρ (similar to Step 2 from sub-section A.5 of appendix A.5)

$$\|Q^* - Q^{\pi_K}\|_\rho^2 \leqslant \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2} \right]^2 \times$$
$$\int \rho(dx, da) \left[\sum_{k=0}^{K-1} \alpha_k A_k \epsilon_k^2 + \alpha_K A_K (Q^* - Q_0)^2 \right] (x,a) \text{ (twice Jensen)}$$

Using assumption 3 (assumption 1 in chapter 2), each term ρA_k is bounded as

$$\begin{aligned} \rho A_k &= \frac{1-\gamma}{2} \rho (I - \gamma P^{\pi_K})^{-1} \left[(P^{\pi^*})^{K-k} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_{k+1}} \right] \\ &= \frac{1-\gamma}{2} \sum_{m \geqslant 0} \gamma^m \rho (P^{\pi_K})^m \left[(P^{\pi^*})^{K-k} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_{k+1}} \right] \\ &\leqslant (1-\gamma) \sum_{m \geqslant 0} \gamma^m \beta_\mu (m + K - k) \mu \qquad \text{(def A.4.3)} \end{aligned}$$

Thus

$$\begin{aligned}
&\|Q^* - Q^{\pi_K}\|_\rho^2 \leqslant \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \times \qquad \text{(A.39)}\\
&\left[\frac{1}{1-\gamma^{K+1}}\sum_{k=0}^{K-1}(1-\gamma)^2\sum_{m\geqslant 0}\gamma^{m+K-k-1}\beta_\mu(m+K-k)\,\|\epsilon_k\|_\mu^2 + \alpha_K(2\bar{C})^2\right]\\
&\leqslant \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2\left[\frac{1}{1-\gamma^{K+1}}\beta_\mu\max_k\|\epsilon_k\|_\mu^2 + \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}}(2\bar{C})^2\right]\\
&\qquad\text{(assumption 3)}\\
&\leqslant \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2\left[\frac{1}{1-\gamma^{K+1}}\beta_\mu\max_k\|\epsilon_k\|_\mu^2 + \frac{\gamma^K}{1-\gamma^{K+1}}(2\bar{C})^2\right]\\
&\leqslant \left[\frac{2\gamma}{(1-\gamma)^2}\right]^2\left[\beta_\mu\max_k\|\epsilon_k\|_\mu^2 + \gamma^K(2\bar{C})^2\right]
\end{aligned}$$

Using $a^2+b^2 \leqslant (a+b)^2$ for nonnegative a, b, we thus conclude that

$$\|Q^* - Q^{\pi_K}\|_\rho \leqslant \frac{2\gamma}{(1-\gamma)^2}\left(\sqrt{\beta_\mu}\max_k\|\epsilon_k\|_\mu + 2\gamma^{K/2}\bar{C}\right) \qquad \text{(A.40)}$$

Step 5: Bounding $C^* - C^{\pi_K}$ Using the performance difference lemma (lemma 6.1 of (Kakade and Langford, 2002), which states that $C^* - C^{\pi_K} = -\frac{1}{1-\gamma}\mathbb{E}_{\substack{x\sim d_{\pi_K}\\ a\sim\pi_K}} A^*[x,a]$. We can upper-bound the performance difference of value function as

$$\begin{aligned}
&C^* - C^{\pi_K} = \frac{1}{1-\gamma}\mathbb{E}_{\substack{x\sim d_{\pi_K}\\ a\sim\pi_K}}[C^*(x) - Q^*(x,a)]\\
&= \frac{1}{1-\gamma}\mathbb{E}_{x\sim d_{\pi_K}}[C^*(x) - Q^*(x,\pi_K(x))]\\
&\leqslant \frac{1}{1-\gamma}\mathbb{E}_{x\sim d_{\pi_K}}[Q^*(x,\pi^*(x)) - Q_K(x,\pi^*(x)) + Q_K(x,\pi_K(x) - Q^*(x,\pi_K(x))]\\
&\qquad\text{(greedy)}\\
&\leqslant \frac{1}{1-\gamma}\mathbb{E}_{x\sim d_{\pi_K}}|Q^*(x,\pi^*(x)) - Q_K(x,\pi^*(x))| + |Q_K(x,\pi_K(x) - Q^*(x,\pi_K(x))|\\
&\leqslant \frac{1}{1-\gamma}\left(\|Q^* - Q^{\pi_K}\|_{d_{\pi_K}\times\pi^*} + \|Q^* - Q^{\pi_K}\|_{d_{\pi_K}\times\pi_K}\right)\\
&\qquad\text{(upper-bound 1-norm by 2-norm)}\\
&\leqslant \frac{2\gamma}{(1-\gamma)^3}\left(\sqrt{\beta_\mu}\max_k\|\epsilon_k\|_\mu + 2\gamma^{K/2}\bar{C}\right) \qquad \text{(A.41)}
\end{aligned}$$

Note that inequality (A.41) follows from (A.40) by specifying $\rho = \chi P^{\pi_K}P^{\pi^*}$ and $\rho = \chi P^{\pi_K}P^{\pi_K}$, respectively ($\chi$ is the initial state distribution).

Finite-sample guarantees for Fitted Q Iteration

From (A.36) we have:

$$\begin{aligned}
&\mathbf{P}\Big\{ \|Q_k - \mathbb{T}Q_{k-1}\|_\mu - 2 \inf_{f\in\mathcal{F}} \|f - \mathbb{T}Q_{k-1}\|_\mu > \epsilon \Big\} \qquad \text{(A.42)}\\
&\leqslant 14 \cdot e \cdot (\dim_{\mathcal{F}} + 1) \left(\frac{640\bar{C}^2}{\epsilon^2}\right)^{\dim_{\mathcal{F}}} \cdot \exp\left(-\frac{n\epsilon^2}{48\cdot 214\bar{C}^4}\right)\\
&\quad + \exp\left(-\frac{3}{416}\cdot\frac{n\epsilon^2}{\bar{C}^2}\right)
\end{aligned}$$

Note that $\inf_{f\in\mathcal{F}} \|f - \mathbb{T}Q_{k-1}\|_\mu \leqslant \sup_{h\in\mathcal{F}} \inf_{f\in\mathcal{F}} \|f - \mathbb{T}h\|_\mu = d_{\mathcal{F}}$ (the *inherent Bellman error* from equation A.16). Combining with equation (A.41), we have the conclusion that for any $\epsilon > 0, 0 < \delta < 1$, after K iterations of Fitted Q Iteration, and for π_K the greedy policy wrt Q_K:

$$C^* - C^\pi_K \leqslant \frac{2\gamma}{(1-\gamma)^3}\left(\sqrt{\beta_\mu}(2d_{\mathcal{F}} + \epsilon) + 2\gamma^{K/2}\bar{C}\right)$$

holds with probability $1-\delta$ when $n = O\big(\frac{\bar{C}^4}{\epsilon^2}(\log\frac{K}{\delta} + \dim_{\mathcal{F}}\log\frac{\bar{C}^2}{\epsilon^2} + \log\dim_{\mathcal{F}})\big)$.

Note that compared to the Fitted Value Iteration analysis of (Munos and Szepesvári, 2008), our error includes an extra factor 2 for $d_{\mathcal{F}}$.

Statement for the Bellman-realizable Case

To facilitate the end-to-end generalization analysis of theorem 2.4.4 in chapter 2, we include a version of FQI analysis under Bellman-realizable assumption in this section. The theorem is a consequence of previous analysis in this section.

Assumption 5 (Bellman evaluation realizability)**.** *We consider function classes $\mathcal{F}$ sufficiently rich so that $\forall f, \mathbb{T}f \in \mathcal{F}$.*

Theorem A.6.2 (Guarantee for FQI - Bellman-realizable Case)**.** *Under Assumption 3 and 5, for any $\epsilon > 0, \delta \in (0,1)$, after K iterations of Fitted Q Iteration, for $n \geqslant \frac{24\cdot 214\cdot\bar{C}^4}{\epsilon^2}\big(\log\frac{K}{\delta} + \dim_{\mathcal{F}}\log\frac{320\bar{C}^2}{\epsilon^2} + \log(14e(\dim_{\mathcal{F}}+1))\big)$, we have with probability $1-\delta$:*

$$C^* - C(\pi_K) \leqslant \frac{2\gamma}{(1-\gamma)^3}\big(\sqrt{\beta_\mu}\epsilon + 2\gamma^{K/2}\bar{C}\big)$$

where π_K is the policy greedy with respect to the returned function Q_K, and C^* is the value of the optimal policy.

A.7 Additional Instantiation of Meta-Algorithm

We provide an additional instantiation of the meta-algorithm described in chapter 2, with Online Gradient Descent (OGD) (Zinkevich, 2003) and Least-Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003b) as subroutines. Using LSPI requires a feature map ϕ such that any state-action pair can be represented by k features. The value function is linear in parameters represented by ϕ. Policy representation is simplified to a weight vector $w \in \mathbb{R}^k$.

Similar to our main algorithm 2, OGD updates require bounded parameters λ. We thus introduce hyper-parameter B as the bound of λ in ℓ_2 norm. The gradient update is projected to the ℓ_2 ball when the norm of λ exceeds B (line 15 of algo 16).

Algorithm 16 Batch Learning under Constraints using Online Gradient Descent and Least-Squares Policy Iteration

Input: Dataset $\mathrm{D} = \{x_i, a_i, x'_i, c_i, g_i\}_{i=1}^n \sim \pi_{\mathrm{D}}$. Online algorithm parameters: ℓ_2 norm bound B, learning rate η

Input: Number of basis function k. Basis function ϕ (feature map for state-action pairs)

1: Initialize $\lambda_1 = (0, \ldots, 0) \in \mathbb{R}^m$
2: **for** each round t **do**
3: Learn $w_t \leftarrow \texttt{LSPI}(c + \lambda_t^\top g)$ *// LSPI with cost $c + \lambda_t^\top g$*
4: Evaluate $\widehat{C}(w_t) \leftarrow \texttt{LSTDQ}(w_t, c)$ *// Algo 18 with π_t, cost c*
5: Evaluate $\widehat{G}(w_t) \leftarrow \texttt{LSTDQ}(w_t, g)$ *// Algo 18 with π_t,* cost g
6: $\widehat{w}_t \leftarrow \frac{1}{t}\sum_{t'=1}^t w_{t'}$
7: $\widehat{C}(\widehat{w}_t) \leftarrow \frac{1}{t}\sum_{t'=1}^t \widehat{C}(w_{t'})$, $\widehat{G}(\widehat{w}_t) \leftarrow \frac{1}{t}\sum_{t'=1}^t \widehat{G}(w_{t'})$
8: $\widehat{\lambda}_t \leftarrow \frac{1}{t}\sum_{t'=1}^t \lambda_{t'}$
9: Learn $\widetilde{w} \leftarrow \texttt{LSPI}(c + \widehat{\lambda}_t^\top g)$ *// LSPI with cost $c + \widehat{\lambda}_t^\top g$*
10: Evaluate $\widehat{C}(\widetilde{w}) \leftarrow \texttt{LSTDQ}(\widetilde{w}, c)$, $\widehat{G}(\widetilde{w}) \leftarrow \texttt{LSTDQ}(\widetilde{w}, g)$
11: $\widehat{\mathrm{L}}_{\max} = \max_{\lambda, \|\lambda\|_2 \leqslant B} \left(\widehat{C}(\widehat{w}_t) + \lambda^\top(\widehat{G}(\widehat{w}_t) - \tau)\right)$
12: $\widehat{\mathrm{L}}_{\min} = \widehat{C}(\widetilde{w}) + \widehat{\lambda}_t^\top(\widehat{G}(\widetilde{w}) - \tau)$
13: **if** $\widehat{\mathrm{L}}_{\max} - \widehat{\mathrm{L}}_{\min} \leqslant \omega$ **then**
14: Return $\widehat{\pi}_t$ greedy w.r.t $\widehat{w}_t$ $\left(\text{i.e., } \widehat{\pi}_t(x) = \arg\min_{a \in \mathbf{A}} \widehat{w}_t^\top \phi(x, a) \ \ \forall x\right)$
15: **end if**
16: $\lambda_{t+1} = \mathcal{P}(\lambda_t - \eta(\widehat{G}(\pi_t) - \tau))$ where projection $\mathcal{P}(\lambda) = B\frac{\lambda}{\max\{B, \|\lambda\|_2\}}$
17: **end for**

Algorithm 17 Least-Squares Policy Iteration: $\texttt{LSPI}(c)$ (Lagoudakis and Parr, 2003b)

Input: Stopping criterion ϵ

Initialize $w' \leftarrow w_0$
repeat
  $w \leftarrow w'$
  $w' \leftarrow \texttt{LSTDQ}(w, c)$
until $\|w - w'\| \leqslant \epsilon$

Output: Policy weight w $\big($i.e., $\pi(x) = \arg\min_{a \in \mathbf{A}} w^\top \phi(x,a) \ \ \forall x\big)$

Algorithm 18 $\texttt{LSTDQ}(w, c)$ (Lagoudakis and Parr, 2003b)

Initialize $\tilde{\mathbf{A}} \leftarrow \mathbf{0}$ // $k \times k$ matrix
Initialize $\tilde{b} \leftarrow \mathbf{0}$ // $k \times 1$ vector
for each $(x, a, x', c) \in \mathrm{D}$ **do**
  $a' = \arg\min_{\tilde{a} \in \mathbf{A}} w^\top \phi(x', \tilde{a})$
  $\tilde{\mathbf{A}} \leftarrow \tilde{\mathbf{A}} + \phi(x,a)\big(\phi(x,a) - \gamma\phi(x',a')\big)^\top$
  $\tilde{b} \leftarrow \tilde{b} + \phi(x,a)c$
end for
$\tilde{w} \leftarrow \tilde{\mathbf{A}}^{-1}\tilde{b}$

Output: $\tilde{w}$

A.8 Additional Experimental Details

Environment Descriptions and Procedures

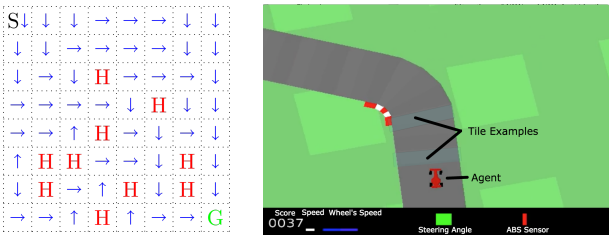


Figure A.1: Depicting the *FrozenLake* and *CarRacing* environments.

Frozen Lake. The environment is a 8x8 grid as seen in Figure A.1 (left), based on OpenAi's FrozenLake-v0. In each episode, the agent starts from S and traverse to goal G. While traversing the grid, the agent must avoid the pre-determined holes denoted by H. If the agent steps off of the grid, the agent returns to the same grid location. The episode terminates when the agent reaches the goal or falls into a hole. The arrows in Figure A.1 (left) is an example policy returned by our algorithm, showing an optimal route.

Denote $\mathbf{X}_{holes}$ as the set of all holes in the grid and $\mathbf{X}_{goal} = \{x_{goal}\}$ is a singleton set representing the goal in the grid. The contrained batch policy learning problem is:

$$\begin{aligned} \min_{\pi \in \Pi} \quad & C(\pi) = \mathbb{E}[\mathbb{I}(x' \notin \mathbf{X}_{goals})] = \mathrm{P}(x' \notin \{x_{goal}\}) \\ \text{s.t.} \quad & G(\pi) = \mathbb{E}[\mathbb{I}(x' \in \mathbf{X}_{holes})] = \mathrm{P}(x' \in \mathbf{X}_{holes}) \leqslant \tau \end{aligned} \tag{A.43}$$

We collect 5000 trajectories by selecting an action randomly with probability .95 and an action from a DDQN-trained model with probability .05.Furthermore we set $B = 30$ and $\eta = 50$, the hyperparameters of our Exponentiated Gradient subroutine. We set the threshold for the constraint $\tau = .1$.

Car Racing. The environment is a racetrack as seen in Figure A.1 (right), modified from OpenAi's CarRacing-v0. In each state, given by the raw pixels, the agent has 12 actions: $a \in A = \{(i, j, k) | i \in \{-1, 0, 1\}, j \in \{0, 1\}, k \in \{0, .2\}\}$. The action tuple (i, j, k) cooresponds to steering angle, amount of gas applied and amount of brake applied, respectively. In each episode, the agent starts at the same point on the track and must traverse over 95% of the track, given by a discretization of 281 tiles. The agent recieves a reward of $+\frac{1000}{281}$ for each unique tile over which the agent drives. The agent receives a penalty of $-.1$ per-time step. Our collected

dataset takes the form: $\mathrm{D} = \{(x_{t-6}, x_{t-3}, x_t), a_t, (x_{t-3}, x_t, x_{t+3}), c_t, g_{0,t}, g_{1,t}\}$ where x_i denotes the image at timestep i and a_t is applied 3 times between x_t and x_{t+3}. This frame-stacking option is common practice in online RL for Atari and video games.In our collected dataset D, the maximum horizon is 469 time steps.

The first constraint concerns accumulated number of brakes, a proxy for smooth driving or acceleration. The second constraint concerns how far the agent travels away from the center of the track, given by the Euclidean distance between the agent and the closest point on the center of the track. Let N_t be the number of tiles that is collected by the agent in time t. The constrained batch policy learning problem is:

$$\begin{aligned} \min_{\pi \in \Pi} \quad & \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t (-\frac{1000}{281} N_t + .1)] \\ \text{s.t.} \quad & G_0(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(a_t \in \mathbf{A}_{braking})] \leqslant \tau_0 \\ & G_1(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t d(u_t, v_t)] \leqslant \tau_1 \end{aligned} \tag{A.44}$$

We instatiate our subroutines, FQE and FQI, with multi-layered CNNs. Furthermore we set $B = 10$ and $\eta = .01$, the hyperparameters of our Exponentiated Gradient subroutine. We set the threshold for the constraint to be about 75% of the value exhibited by online RL agent trained by DDQN (Van Hasselt et al., 2016).

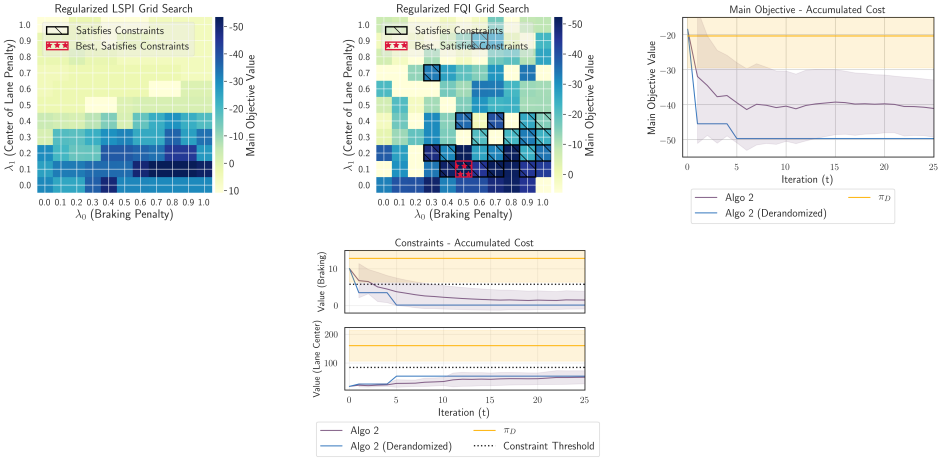


Figure A.2: (First and Second figures) Result of 2-D grid-search for one-shot, regularized policy learning for *LSPI* (left) and *FQI* (right). (Third and Fourth figures) value range of individual policies in our mixtured policy and data generating policy π_D for main objective (left) and cost constraint (right)

Additional Discussion for the Car Racing Experiment

Regularized policy learning and grid-search. We perform grid search over a range of regularization parameters λ for both Least-Squares Policy Iteration - LSPI ((Lagoudakis and Parr, 2003b)) and Fitted Q Iteration - FQI ((Ernst et al., 2005)). The results, seen from the the first and second plot of Figure A.2, show that one-shot regularized learning has difficulty learning a policy that satisfies both constraints. We augment LSPI with non-linear feature mapping from one of our best performing FQI model (using CNNs representation). While both regularized LSPI and regularized FQI can achieve low main objective cost, the constraint cost values tend to be sensitive with the λ step. Overall for the whole grid search, about 10% of regularized policies satisfy both constraints, while none of the regularized LSPI policy satisfies both constraints.

Mixture policy and de-randomization. As our algorithm returned a mixture policy, it is natural to analyze the performance of individual policies in the mixture. The third and fourth plot from Figure A.2 show the range of performance of individual policy in our mixture (purple band). We compare individual policy return with the stochastic behavior of the data generation policy. Note that our policies satisfy constraints almost always, while the individual policy returned in the mixture also tends to outperform π_D with respect to the main objective cost.

Off-policy evaluation standalone comparison. Typically, inverse propensity scoring based methods call for stochastic behavior and evaluation policies (Precup et al., 2000; Swaminathan and Joachims, 2015). However in this domain, the evaluation policy and environment are both deterministic, with long horizon (the max horizon is D is 469). Consequently Per-Decision Importance Sampling typically evaluates the policy as 0. In general, off-policy policy evaluation in long-horizon domains is known to be challenging (Liu et al., 2018c; Guo et al., 2017). We augment PDIS by approximating the evaluation policy with a stochastic policy, using a softmin temperature parameter. However, PDIS still largely shows significant errors. For Doubly Robust and Weighted Doubly Robust methods, we train a model of the environment as follows:

- a 32-dimensional representation of state input is learned using variational autoencoder. Dimensionality reduction is necessary to aid accuracy, as original state dimension is $96 \times 96 \times 3$
- an LSTM is used to learn the transition dynamics $P(z(x')|z(x), a)$, where $z(x)$

is the low-dimensional representation learned from previous step. Technically, using a recurrent neural networks is an augmentation to the dynamical modeling, as true MDPs typically do not require long-term memory

- the model is trained separately on a different dataset, collected indendently from the dataset D used for evaluation

The architecture of our dynamics model is inspired by recent work in model-based online policy learning (Ha and Schmidhuber, 2018). However, despite our best effort, learning the dynamics model accurately proves highly challenging, as the horizon and dimensionality of this domain are much larger than popular benchmarks in the OPE literature (Jiang and Li, 2016a; Thomas and Brunskill, 2016b; Farajtabar et al., 2018a). The dynamics model has difficulty predicting the future state several time steps away. Thus we find that the long-horizon, model-based estimation component of DR and WDR in this high-dimensional setting is not sufficiently accurate. For future work, a thorough benchmarking of off-policy evaluation methods in high-dimensional domains would be a valuable contribution.

Appendix B

APPENDIX TO CHAPTER 3

B.1 Notations

Table B.1: Glossary of terms for Chapter 3

Acronym	Term
X, A, P, R	State, Action, Dynamics, Reward
γ	Discount Factor
d_0	Initial State Distribution
D	Dataset
τ	Trajectory/Episode
T	Horizon/Episode Length
N	Number of episodes in D
π_b, π_e	Behavior Policy, Evaluation Policy
V, Q	Value and Action-Value Function, ex: $V(\pi_e), Q(x, \pi_e)$
$\rho^i_{j:j'}$	Cumulative Importance Weight, $\prod_{t=j}^{\min(j',T-1)} \frac{\pi_e(a^i_t \vert x^i_t)}{\pi_b(a^i_t \vert x^i_t)}$
IPS	Inverse Propensity Scoring
DM	Direct Method
HM	Hybrid Method
IS	Importance Sampling
PDIS	Per-Decision Importance Sampling
WIS	Weighted Importance Sampling
PDWIS	Per-Decision Weighted Importance Sampling
PDWIS	Per-Decision Weighted Importance Sampling
FQE	Fitted Q Evaluation (Le et al., 2019a)
IH	Infinite Horizon (Liu et al., 2018b)
Q-Reg	Q Regression (Farajtabar et al., 2018b)
MRDR	More Robust Doubly Robst (Farajtabar et al., 2018b)
AM	Approximate Model (Model Based)
$Q(\lambda)$	$Q^\pi(\lambda)$ (Harutyunyan et al., 2016)
$R(\lambda)$	Retrace(λ) (Munos et al., 2016)
Tree	Tree-Backup(λ) (Precup et al., 2000)
DR	Doubly-Robust (Jiang and Li, 2016b; Dudík et al., 2011a)
WDR	Weighted Doubly-Robust (Dudík et al., 2011a)
MAGIC	(Thomas and Brunskill, 2016a)
Graph	Graph Environment
Graph-MC	Graph Mountain Car Environment
MC	Mountain Car Environment
Pix-MC	Pixel-Based Mountain Car Environment
Enduro	Enduro Environment
Graph-POMDP	Graph-POMDP Environment
GW	Gridworld Environment
Pix-GW	Pixel-Based Gridworld Environment

B.2 Ranking of Methods

A method that is within 10% of the method with the lowest Relative MSE is counted as a top method, called Near-top Frequency, and then we aggregate across all experiments. See Table B.2 for a sorted list of how often the methods appear within 10% of the best method.

Table B.2: Fraction of time among the top estimators across all experiments

Method	Near-top Frequency
MAGIC FQE	0.300211
DM FQE	0.236786
IH	0.190275
WDR FQE	0.177590
MAGIC $Q^\pi(\lambda)$	0.173362
WDR $Q^\pi(\lambda)$	0.173362
DM $Q^\pi(\lambda)$	0.150106
DR $Q^\pi(\lambda)$	0.135307
WDR R(λ)	0.133192
DR FQE	0.128964
MAGIC R(λ)	0.107822
WDR Tree	0.105708
DR R(λ)	0.105708
DM R(λ)	0.097252
DM Tree	0.084567
MAGIC Tree	0.076110
DR Tree	0.073996
DR MRDR	0.073996
WDR Q-Reg	0.071882
DM AM	0.065539
IS	0.063425
WDR MRDR	0.054968
PDWIS	0.046512
DR Q-Reg	0.044397
MAGIC AM	0.038055
MAGIC MRDR	0.033827
DM MRDR	0.033827
PDIS	0.033827
MAGIC Q-Reg	0.027484
WIS	0.025370
NAIVE	0.025370
DM Q-Reg	0.019027
DR AM	0.012685
WDR AM	0.006342

Empirical Support for Decision Tree Guideline

Tables B.3-B.10 provide a numerical support for the decision tree in the main paper (Figure 3.2). Each table refers to a child node in the decision tree, ordered from

left to right, respectively. For example, Table B.3 refers to the left-most child node (propery specified, short horizon, small policy mismatch) while Table B.10 refers to the right-most child node (misspecified, good representation, long horizon, good π_b estimate).

Table B.3: Near-top Frequency among the properly specified, short horizon, small policy mismatch experiments

	DM	Hybrid			IPS	
	Direct	DR	WDR	MAGIC	Standard	Per-Decision
AM	4.7%	4.7%	3.1%	4.7%		
Q-Reg	0.0%	4.7%	6.2%	4.7%		
MRDR	7.8%	14.1%	7.8%	7.8%		
FQE	**40.6**%	23.4%	21.9%	**34.4**%		
R(λ)	17.2%	20.3%	20.3%	14.1%		
Q$^\pi$(λ)	21.9%	18.8%	18.8%	17.2%		
Tree	15.6%	12.5%	12.5%	14.1%		
IH	17.2%	-	-	-		
IS					**4.7**%	4.7%
WIS					3.1%	3.1%
NAIVE					1.6%	-

Table B.4: Near-top Frequency among the properly specified, short horizon, large policy mismatch experiments

	DM	Hybrid			IPS	
	Direct	DR	WDR	MAGIC	Standard	Per-Decision
AM	20.3%	1.6%	0.0%	7.8%		
Q-Reg	1.6%	1.6%	3.1%	1.6%		
MRDR	3.1%	1.6%	6.2%	1.6%		
FQE	**35.9**%	14.1%	17.2%	**37.5**%		
R(λ)	23.4%	14.1%	20.3%	23.4%		
Q$^\pi$(λ)	15.6%	15.6%	14.1%	20.3%		
Tree	21.9%	12.5%	18.8%	21.9%		
IH	29.7%	-	-	-		
IS					0.0%	0.0%
WIS					0.0%	**1.6**%
NAIVE					3.1%	-

Table B.5: Near-top Frequency among the properly specified, long horizon, small policy mismatch experiments

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	6.9%	0.0%	0.0%	5.6%
Q-Reg	0.0%	1.4%	1.4%	1.4%
MRDR	1.4%	0.0%	1.4%	2.8%
FQE	**50.0**%	22.2%	23.6%	**50.0**%
R(λ)	13.9%	12.5%	11.1%	9.7%
Q$^{\pi}$(λ)	20.8%	18.1%	18.1%	18.1%
Tree	2.8%	1.4%	0.0%	2.8%
IH	29.2%	-	-	-

	IPS	
	Standard	Per-Decision
IS	**0.0**%	0.0%
WIS	0.0%	0.0%
NAIVE	5.6%	-

Table B.6: Near-top Frequency among the properly specified, long horizon, large policy mismatch, deterministic env/rew experiments

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	3.5%	3.5%	1.8%	1.8%
Q-Reg	3.5%	1.8%	0.0%	0.0%
MRDR	3.5%	1.8%	0.0%	0.0%
FQE	15.8%	17.5%	29.8%	28.1%
R(λ)	1.8%	3.5%	0.0%	0.0%
Q$^{\pi}$(λ)	**22.8**%	15.8%	**38.6**%	24.6%
Tree	3.5%	3.5%	1.8%	1.8%
IH	21.1%	-	-	-

	IPS	
	Standard	Per-Decision
IS	5.3%	3.5%
WIS	0.0%	**8.8**%
NAIVE	0.0%	-

Table B.7: Near-top Frequency among the properly specified, long horizon, large policy mismatch, stochastic env/rew experiments

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	14.6%	0.0%	0.0%	8.3%
Q-Reg	4.2%	2.1%	0.0%	2.1%
MRDR	4.2%	2.1%	0.0%	0.0%
FQE	31.2%	2.1%	0.0%	**25.0**%
R(λ)	4.2%	6.2%	0.0%	0.0%
Q$^{\pi}$(λ)	2.1%	0.0%	0.0%	2.1%
Tree	4.2%	6.2%	0.0%	0.0%
IH	**41.7**%	-	-	-

	IPS	
	Standard	Per-Decision
IS	**25.0**%	4.2%
WIS	0.0%	0.0%
NAIVE	2.1%	-

Table B.8: Near-top Frequency among the potentially misspecified, insufficient representation experiments

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	-	-	-	-
Q-Reg	3.9%	13.7%	**25.5**%	6.9%
MRDR	0.0%	18.6%	15.7%	5.9%
FQE	0.0%	5.9%	13.7%	24.5%
R(λ)	-	-	-	-
Q$^{\pi}$(λ)	-	-	-	-
Tree	-	-	-	-
IH	**6.9**%	-	-	-

	IPS	
	Standard	Per-Decision
IS	10.8%	8.8%
WIS	9.8%	**13.7**%
NAIVE	3.9%	-

Table B.9: Near-top Frequency among the potentially misspecified, sufficient representation, poor π_b estimate experiments

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	0.0%	0.0%	0.0%	0.0%
Q-Reg	0.0%	0.0%	3.3%	0.0%
MRDR	13.3%	6.7%	0.0%	0.0%
FQE	0.0%	3.3%	6.7%	10.0%
R(λ)	16.7%	0.0%	6.7%	**20.0**%
Q$^\pi$(λ)	6.7%	0.0%	0.0%	3.3%
Tree	**20.0**%	0.0%	6.7%	6.7%
IH	0.0%	-	-	-

	IPS	
	Standard	Per-Decision
IS	**3.3**%	0.0%
WIS	0.0%	0.0%
NAIVE	0.0%	-

Table B.10: Near-top Frequency among the potentially misspecified, sufficient representation, good π_b estimate experiments

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	0.0%	0.0%	0.0%	2.8%
Q-Reg	0.0%	0.0%	0.0%	0.0%
MRDR	0.0%	5.6%	0.0%	5.6%
FQE	**8.3**%	8.3%	**25.0**%	11.1%
R(λ)	2.8%	8.3%	8.3%	19.4%
Q$^\pi$(λ)	5.6%	5.6%	8.3%	0.0%
Tree	5.6%	8.3%	16.7%	5.6%
IH	0.0%	-	-	-

	IPS	
	Standard	Per-Decision
IS	**0.0**%	0.0%
WIS	0.0%	0.0%
NAIVE	0.0%	-

B.3 Challenging Common Wisdom - Supporting Data

The following tables represent the numerical support for how horizon and policy difference affect the performance of the OPE estimators when policy mismatch is held constant. Notice that the policy mismatch for table B.12 and B.13 are identical: $\left(\frac{.124573...}{.1}\right)^{100} = \left(\frac{.9}{.1}\right)^{10}$. What we see here is that despite identical policy mismatch, the longer horizon does not impact the error as much (compared to the baseline, Table B.11) as moving π_e to .9, far from .1 and keeping the horizon the same.

Table B.11: Graph, relative MSE. $T = 10, N = 50, \pi_b(a = 0) = 0.1, \pi_e(a = 0) = 0.1246$. Dense rewards. *Baseline*.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	1.9E-3	4.9E-3	5.0E-3	3.4E-3
Q-Reg	2.4E-3	4.3E-3	4.2E-3	4.5E-3
MRDR	5.8E-3	8.9E-3	9.4E-3	9.2E-3
FQE	1.8E-3	1.8E-3	1.8E-3	1.8E-3
R(λ)	1.8E-3	1.8E-3	1.8E-3	**1.8E-3**
Q$^\pi(\lambda)$	1.8E-3	1.8E-3	1.8E-3	1.8E-3
Tree	1.8E-3	1.8E-3	1.8E-3	1.8E-3
IH	**1.6E-3**	-	-	-

	IPS	
	Standard	Per-Decision
IS	**5.6E-4**	8.4E-4
WIS	1.4E-3	1.4E-3
NAIVE	6.1E-3	-

Table B.12: Graph, relative MSE. $T = 100, N = 50, \pi_b(a = 0) = 0.1, \pi_e(a = 0) = 0.1246$. Dense rewards. *Increasing horizon compared to baseline, fixed π_e.*

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	5.6E-2	5.9E-2	5.9E-2	5.3E-2
Q-Reg	3.4E-3	1.1E-1	1.2E-1	9.2E-2
MRDR	1.1E-2	2.5E-1	2.9E-1	3.1E-1
FQE	6.0E-2	6.0E-2	6.0E-2	6.0E-2
R(λ)	6.0E-2	6.0E-2	6.0E-2	6.0E-2
Q$^\pi(\lambda)$	6.0E-2	6.0E-2	6.0E-2	6.0E-2
Tree	3.4E-1	7.0E-3	**1.6E-3**	2.3E-3
IH	**4.7E-4**	-	-	-

	IPS	
	Standard	Per-Decision
IS	1.7E-2	2.5E-3
WIS	9.5E-4	**4.9E-4**
NAIVE	5.4E-3	-

Table B.13: Graph, relative MSE. $T = 10, N = 50, \pi_b(a = 0) = 0.1, \pi_e(a = 0) = 0.9$. Dense rewards. *Increasing π_e compared to baseline, fixed horizon.*

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	6.6E-1	6.7E-1	6.6E-1	6.6E-1
Q-Reg	5.4E-1	**6.3E-1**	1.3E0	9.3E-1
MRDR	5.4E-1	7.3E-1	2.0E0	2.0E0
FQE	6.6E-1	6.6E-1	6.6E-1	6.6E-1
R(λ)	6.7E-1	6.6E-1	9.3E-1	1.0E0
Q$^{\pi}$(λ)	6.6E-1	6.6E-1	6.6E-1	6.6E-1
Tree	6.7E-1	6.6E-1	9.4E-1	1.0E0
IH	**1.4E-2**	-	-	-

	IPS	
	Standard	Per-Decision
IS	1.0E0	**5.4E-1**
WIS	2.0E0	9.7E-1
NAIVE	4.0E0	-

B.4 Methods

Below we include a description of each of the methods we tested. Let $\tilde{T} = T - 1$.

Inverse Propensity Scoring (IPS) Methods

Table B.14: IPS methods. Dudík et al. (2011b); Jiang and Li (2016a)

	STANDARD	PER-DECISION
IS	$\sum_{i=1}^{N} \frac{\rho^i_{0:\tilde{T}}}{N} \sum_{t=0}^{\tilde{T}} \gamma^t r_t$	$\sum_{i=1}^{N} \sum_{t=0}^{\tilde{T}} \gamma^t \frac{\rho^i_{0:t}}{N} r_t$
WIS	$\sum_{i=1}^{N} \frac{\rho^i_{0:\tilde{T}}}{w_{0:\tilde{T}}} \sum_{t=0}^{\tilde{T}} \gamma^t r_t$	$\sum_{i=1}^{N} \sum_{t=0}^{\tilde{T}} \gamma^t \frac{\rho^i_{0:t}}{w_{0:t}} r_t$

Table B.14 shows the calculation for the four traditional IPS estimators: V_{IS}, V_{PDIS}, V_{WIS}, V_{PDWIS}. In addition, we include the following method as well since it is a Rao-Blackwellization Liu et al. (2018b) of the IPS estimators:

Hybrid Methods

Hybrid rely on being supplied an action-value function $\widehat{Q}$, an estimate of Q, from which one can also yield $\widehat{V}(x) = \sum_{a \in A} \pi(a|x)\widehat{Q}(x,a)$. Doubly-Robust (DR): Thomas and Brunskill (2016a); Jiang and Li (2016b)

$$V_{DR} = \frac{1}{N}\sum_{i=1}^{N} \widehat{V}(x^i_0) + \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{\infty} \gamma^t \rho^i_{0:t}[r^i_t - \widehat{Q}(x^i_t, a^i_t) + \gamma \widehat{V}(x^i_{t+1})]$$

Weighted Doubly-Robust (WDR): Thomas and Brunskill (2016a)

$$V_{WDR} = \frac{1}{N}\sum_{i=1}^{N} \widehat{V}(x^i_0) + \sum_{i=1}^{N}\sum_{t=0}^{\infty} \gamma^t \frac{\rho^i_{0:t}}{w_{0:t}}[r^i_t - \widehat{Q}(x^i_t, a^i_t) + \gamma \widehat{V}(x^i_{t+1})]$$

MAGIC: Thomas and Brunskill (2016a) Given $g_J = \{g^i | i \in J \subseteq \mathbb{N} \cup \{-1\}\}$ where

$$g^j(D) = \sum_{i=1}^{N}\sum_{t=0}^{j} \gamma^t \frac{\rho^i_{0:t}}{w_{0:t}} r^i_t + \sum_{i=1}^{N} \gamma^{j+1} \frac{\rho^i_{0:t}}{w_{0:t}} \widehat{V}(x^i_{j+1}) - \sum_{i=1}^{N}\sum_{t=0}^{j} \gamma^t \left(\frac{\rho^i_{0:t}}{w_{0:t}} \widehat{Q}(x^i_t, a^i_t) - \frac{\rho^i_{0:\tilde{T}}}{w_{0:\tilde{T}}} \widehat{V}(x^i_t)\right),$$

then define $dist(y, Z) = \min_{z \in Z} |y - z|$ and

$$\widehat{b}_n(j) = dist(g_j^J(D), CI(g^\infty(D), 0.5))$$
$$\widehat{\Omega}_n(i, j) = Cov(g_i^J(D), g_j^J(D))$$

then, for a $|J|-$simplex $\Delta^{|J|}$ we can calculate

$$\widehat{x}^* \in \arg\min_{x \in \Delta^{|J|}} x^T[\widehat{\Omega}_n + \widehat{b}\widehat{b}^T]x$$

which, finally, yields

$$V_{MAGIC} = (\widehat{x}^*)^T g_J.$$

MAGIC can be thought of as a weighted average of different blends of the DM and Hybrid. In particular, for some $i \in J$, g^i represents estimating the first i steps of $V(\pi_e)$ according to DR (or WDR) and then estimating the remaining steps via $\widehat{Q}$. Hence, V_{MAGIC} finds the most appropriate set of weights which trades off between using a direct method and a Hybrid.

Direct Methods (DM)

Model-Based

Approximate Model (AM): Jiang and Li (2016b) An approach to model-based value estimation is to directly fit the transition dynamics $P(x_{t+1}|x_t, a_t)$, reward $R(x_t, a_t)$, and terminal condition $P(x_{t+1} \in X_{terminal}|x_t, a_t)$ of the MDP using some for of maximum likelihood or function approximation. This yields a simulation environment from which one can extract the value of a policy using an average over rollouts. Thus, $V(\pi) = \mathbb{E}[\sum_{t=1}^T \gamma^t r(x_t, a_t)|x_0 = x, a_0 = \pi(x_0)]$ where the expectation is over initial conditions $x \sim d_0$ and the transition dynamics of the simulator.

Model-Free

Every estimator in this section will approximate Q with $\widehat{Q}(\cdot; \theta)$, parametrized by some θ. From $\widehat{Q}$ the OPE estimate we seek is

$$V = \frac{1}{N} \sum_{i=1}^N \sum_{a \in A} \pi_e(a|s) \widehat{Q}(s_0^i, a; \theta)$$

Note that $\mathbb{E}_{\pi_e} Q(x_{t+1}, \cdot) = \sum_{a \in A} \pi_e(a|x_{t+1}) Q(x_{t+1}, a)$.

Direct Model Regression (Q-Reg): Farajtabar et al. (2018b)

$$\widehat{Q}(\cdot,\theta) = \min_{\theta} \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{\tilde{T}} \gamma^t \rho^i_{0:t}\left(R^i_{t:\tilde{T}} - \widehat{Q}(x^i_t, a^i_t;\theta)\right)^2$$

$$R^i_{t:\tilde{T}} = \sum_{t'=t}^{\tilde{T}} \gamma^{t'-t}\rho^i_{(t+1):t'} r^i_{t'}$$

Fitted Q Evaluation (FQE): Le et al. (2019a) $\widehat{Q}(\cdot,\theta) = \lim_{k\to\infty}\widehat{Q}_k$ where

$$\widehat{Q}_k = \min_{\theta}\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{\tilde{T}}(\widehat{Q}_{k-1}(x^i_t, a^i_t;\theta) - y^i_t)^2$$

$$y^i_t \equiv r^i_t + \gamma \mathbb{E}_{\pi_e}\widehat{Q}_{k-1}(x^i_{t+1}, \cdot;\theta)$$

Retrace(λ) (R(λ)), Tree-Backup (Tree), $Q^\pi(\lambda)$: Munos et al. (2016); Precup et al. (2000); Harutyunyan et al. (2016) $\widehat{Q}(\cdot,\theta) = \lim_{k\to\infty}\widehat{Q}_k$ where

$$\widehat{Q}_k(x,a;\theta) = \widehat{Q}_{k-1}(x,a;\theta) + \mathbb{E}_{\pi_b}[\sum_{t\geqslant 0}\gamma^t \prod_{s=1}^{t} c_s y_t | x_0 = x, a_0 = a]$$

and

$$y_t = r^t + \gamma \mathbb{E}_{\pi_e}\widehat{Q}_{k-1}(x_{t+1}, \cdot;\theta) - \widehat{Q}_{k-1}(x_t, a_t;\theta)$$

$$c_s = \begin{cases} \lambda \min(1, \frac{\pi_e(a_s|x_s)}{\pi_b(a_s|x_s)}) & R(\lambda) \\ \lambda \pi_e(a_s|x_s) & Tree \\ \lambda & Q^\pi(\lambda) \end{cases}$$

More Robust Doubly-Robust (MRDR): Farajtabar et al. (2018b) Given

$$\Omega_{\pi_b}(x) = diag[1/\pi_b(a|x)]_{a\in A} - ee^T$$

$$e = [1,\ldots,1]^T$$

$$R^i_{t:\tilde{T}} = \sum_{j=t}^{\tilde{T}} \gamma^{j-t}\rho^i_{(t+1):j} r(x^i_j, a^i_j)$$

and

$$q_\theta(x,a,r) = diag[\pi_e(a'|x)]_{a'\in A}[\widehat{Q}(x,a';\theta)]_{a'\in A} - r[\mathbf{1}\{a'=a\}]_{a'\in A}$$

where **1** is the indicator function, then

$$\widehat{Q}(\cdot,\theta) = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{\tilde{T}} \gamma^{2t} (\rho^i_{0:\tilde{T}})^2 \times \rho^i_t q_\theta(x^i_t, a^i_t, R^i_{t:\tilde{T}})^T \Omega_{\pi_b}(x^i_t) q_\theta(x^i_t, a^i_t, R^i_{t:\tilde{T}})$$

State Density Ratio Estimation (IH): Liu et al. (2018b)

$$V_{IH} = \sum_{i=1}^{N} \sum_{t=0}^{\tilde{T}} \frac{\gamma^t \omega(s^i_t) \rho_{t:t} r^i_t}{\sum_{i'=0}^{N} \sum_{t'=1}^{\tilde{T}} \gamma^{t'} \omega(s^{i'}_{t'}) \rho_{t':t'}}$$

$$\omega(s^i_t) = \lim_{t \to \infty} \frac{\sum_{t=0}^{T} \gamma^t d_{\pi_e}(s^i_t)}{\sum_{t=0}^{T} \gamma^t d_{\pi_b}(s^i_t)}$$

where π_b is assumed to be a fixed data-generating policy, and d_π is the distribution of states when executing π from $s_0 \sim d_0$. The details for how to find ω can be found in Algorithm 1 and 2 of Liu et al. (2018b).

B.5 Environments

For every environment, we initialize the environment with a fixed horizon length T. If the agent reaches a goal before T or if the episode is not over by step T, it will transition to an environment-dependent absorbing state where it will stay until time T. For a high level description of the environment features, see Table 3.1.

Environment Descriptions

Graph

Figure B.1 shows a visualization of the Toy-Graph environment. The graph is initialized with horizon T and with absorbing state $x_{abs} = 2T$. In each episode, the agent starts at a single starting state $x_0 = 0$ and has two actions, $a = 0$ and $a = 1$. At each time step $t < T$, the agent can enter state $x_{t+1} = 2t + 1$ by taking action $a = 0$, or $x_{t+1} = 2t + 2$ by taking action $a = 1$. If the environment is stochastic, we simulate noisy transitions by allowing the agent to slip into $x_{t+1} = 2t + 2$ instead of $x_{t+1} = 2t + 1$ and vice-versa with probability .25. At the final time $t = T$, the agent always enters the terminal state x_{abs}. The reward is $+1$ if the agent transitions to an odd state, otherwise is -1. If the environment provides sparse rewards, then $r = +1$ if x_{T-1} is odd, $r = -1$ if x_{T-1} is even, otherwise $r = 0$. Similarly to deterministic rewards, if the environment's rewards are stochastic, then the reward is $r \sim N(1,1)$ if the agent transitions to an odd state, otherwise $r \sim N(-1,1)$. If the rewards are

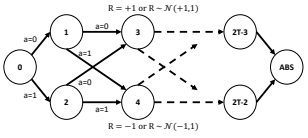


Figure B.1: Graph Environment

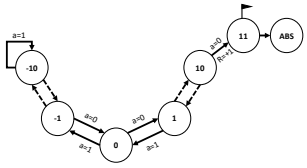


Figure B.2: Graph-MC Environment

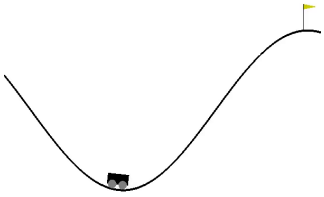


Figure B.3: MC Environment, pixel-version. The non-pixel version involves representing the state of the car as the position and velocity.

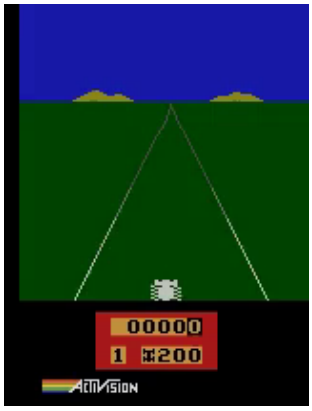


Figure B.4: Enduro Environment

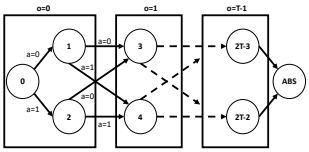


Figure B.5: Graph-POMDP Environment. Model-Fail Thomas and Brunskill (2016a) is a special case of this environment where T=2. We also extend the environment to arbitrary horizon which makes it a semi-mdp.

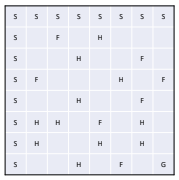


Figure B.6: Gridworld environment. Blank spaces indicate areas of a small negative reward, S indicates the starting states, F indicates a field of slightly less negative reward, H indicates a hole of severe penalty, G indicates the goal of positive reward.

sparse and stochastic then $r \sim N(1,1)$ if x_{T-1} is odd, otherwise $r \sim N(-1,1)$ and $r = 0$ otherwise.

Graph-POMDP

Figure B.5 shows a visualization of the Graph-POMDP environment. The underlying state structure of Graph-POMDP is exactly the Graph environment. However, the states are grouped together based on a choice of Graph-POMDP horizon length, H. This parameter groups states into H observable states. The agent only is able to observe among these states, and not the underlying MDP structure. Model-Fail Thomas and Brunskill (2016a) is a special case of this environment when $H = T = 2$.

Graph Mountain Car (Graph-MC)

Figure B.2 shows a visualization of the Toy-MC environment. This environment is a 1-D graph-based simplification of Mountain Car. The agent starts at $x_0 = 0$, the center of the valley and can go left or right. There are 21 total states, 10 to the left of the starting position and 11 to the right of the starting position, and a terminal absorbing state $x_{abs} = 22$. The agent receives a reward of $r = -1$ at every timestep. The reward becomes zero if the agent reaches the goal, which is state $x = +11$. If the agent reaches $x = -10$ and continues left then the agent remains in $x = -10$. If the agent does not reach state $x = +11$ by step T then the episode terminates and the agent transitions to the absorbing state.

Mountain Car (MC)

We use the OpenAI version of Mountain Car with a few simplifying modifications Brockman et al. (2016); Sutton and Barto (2018a). The car starts in a valley and has to go back and forth to gain enough momentum to scale the mountain and reach the end goal. The state space is given by the position and velocity of the car. At each time step, the car has the following options: accelerate backwards, forwards or do nothing. The reward is $r = -1$ for every time step until the car reaches the goal. While the original trajectory length is capped at 200, we decrease the effective length by applying every action a_t five times before observing x_{t+1}. Furthermore, we modify the random initial position from being uniformly between $[-.6, -.4]$ to being one of $\{-.6, -.5, -.4\}$, with no velocity. The environment is initialized with a horizon T and absorbing state $x_{abs} = [.5, 0]$, position at .5 and no velocity.

Pixel-based Mountain Car (Pix-MC)

This environment is identical to Mountain Car except the state space has been modified from position and velocity to a pixel based representation of a ball, representing a car, rolling on a hill, see Figure B.3. Each frame f_t is a 80×120 image of the ball on the mountain. One cannot deduce velocity from a single frame, so we represent the state as $x_t = \{f_{t-1}, f_t\}$ where $f_{-1} = f_0$, the initial state. Everything else is identical between the pixel-based version and the position-velocity version described earlier.

Enduro

We use OpenAI's implementation of Enduro-v0, an Atari 2600 racing game. We downsample the image to a grayscale of size (84,84). We apply every action one time and we represent the state as $x_t = \{f_{t-3}, f_{t-2}, f_{t-1}, f_t\}$ where $f_i = f_0$, the initial state, for $i < 0$. See Figure B.4 for a visualization.

Gridworld (GW)

Figure B.6 shows a visualization of the Gridworld environment. The agent starts at a state in the first row or column (denoted S in the figure), and proceeds through the grid by taking actions, given by the four cardinal directions, for $T = 25$ timesteps. An agent remains in the same state if it chooses an action which would take it out of the environment. If the agent reaches the goal state G, in the bottom right corner of the environment, it transitions to a terminal state $x = 64$ for the remainder of the trajectory and receives a reward of $+1$. In the grid, there is a field (denoted F) which gives the agent a reward of $-.005$ and holes (denoted H) which give $-.5$. The remaining states give a reward of $-.01$.

Pixel-Gridworld (Pixel-GW)

This environment is identical to Gridworld except the state space has been modified from position to a pixel based representation of the position: 1 for the agent's location, 0 otherwise. We use the same policies as in the Gridworld case.

B.6 Experimental Setup

Description of the policies

Graph, Graph-POMDP and Graph-MC use static policies with some probability of going left and another probability of going right, ex: $\pi(a = 0) = p, \pi(a = 1) = 1 - p$, independent of state. We vary p in our experiments.

GW, Pix-GW, MC, Pixel-MC, and Enduro all use an $\epsilon-$Greedy policy. In other words, we train a policy Q^* (using value iteration or DDQN) and then vary the deviation away from the policy. Hence $\epsilon - Greedy(Q^*)$ implies we follow a mixed policy $\pi = \arg\max_a Q^*(x, a)$ with probability $1 - \epsilon$ and uniform with probability ϵ. We vary ϵ in our experiments.

Enumeration of Experiments

Graph

See Table B.15 for a description of the parameters of the experiment we ran in the Graph Environment. The experiments are the Cartesian product of the table.

Table B.15: Graph parameters

	Parameters
γ	.98
N	$2^{3:11}$
T	$\{4, 16\}$
$\pi_b(a=0)$	$\{.2, .6\}$
$\pi_e(a=0)$	.8
Stochastic Env	{True, False}
Stochastic Rew	{True, False}
Sparse Rew	{True, False}
Seed	{10 of random$(0 : 2^{16})$}
ModelType	Tabular
Regress π_b	False

Graph-POMDP

See Table B.16 for a description of the parameters of the experiment we ran in the Graph-POMDP Environment. The experiments are the Cartesian product of the table.

Table B.16: Graph-POMDP parameters

	Parameters
γ	.98
N	$2^{8:11}$
(T,H)	$\{(2, 2), (16, 6)\}$
$\pi_b(a=0)$	$\{.2, .6\}$
$\pi_e(a=0)$	.8
Stochastic Env	{True, False}
Stochastic Rew	{True, False}
Sparse Rew	{True, False}
Seed	{10 of random$(0 : 2^{16})$}
ModelType	Tabular
Regress π_b	False

Gridworld

See Table B.17 for a description of the parameters of the experiment we ran in the Gridworld Environment. The experiments are the Cartesian product of the table.

Table B.17: Gridworld parameters

	Parameters
γ	.98
N	$2^{6:11}$
T	25
$\epsilon-$ Greedy, π_b	{.2, .4, .6, .8, 1.}
$\epsilon-$ Greedy, π_e	.1
Stochastic Env	False
Stochastic Rew	False
Sparse Rew	False
Seed	{10 of random$(0:2^{16})$}
ModelType	Tabular
Regress π_b	True

Pixel-Gridworld (Pix-GW)

See Table B.18 for a description of the parameters of the experiment we ran in the Pix-GW Environment. The experiments are the Cartesian product of the table.

Table B.18: Pix-GW parameters

	Parameters
γ	.96
N	$2^{6:9}$
T	25
$\epsilon-$ Greedy, π_b	{.2, .4, .6, .8, 1.}
$\epsilon-$ Greedy, π_e	.1
Stochastic Env	{True, False}
Stochastic Rew	False
Sparse Rew	False
Seed	{10 of random$(0:2^{16})$}
ModelType	NN
Regress π_b	{True, False}

Graph-MC

See Table B.19 for a description of the parameters of the experiment we ran in the TMC Environment. The experiments are the Cartesian product of the table.

Table B.19: Graph-MC parameters

	Parameters
γ	.99
N	$2^{7:11}$
T	250
$(\pi_b(a=0), \pi_e(a=0))$	$\{(.45, .45), (.6, .6), (.45.6)$ $(.6, .45), (.8, .2), (.2, .8)\}$
Stochastic Env	False
Stochastic Rew	False
Sparse Rew	False
Seed	{10 of random$(0 : 2^{16})$}
ModelType	Tabular
Regress π_b	False

Mountain Car (MC)

See Table B.20 for a description of the parameters of the experiment we ran in the MC Environment. The experiments are the Cartesian product of the table.

Table B.20: MC parameters

	Parameters
γ	.99
N	$2^{7:10}$
T	250
$\epsilon-$ Greedy, (π_b, π_e)	$\{(.1, 0), (1, 0)$ $(1, .1), (.1, 1)\}$
Stochastic Env	False
Stochastic Rew	False
Sparse Rew	False
Seed	{10 of random$(0 : 2^{16})$}
ModelType	{Tabular, NN}
Regress π_b	False

Pixel-Mountain Car (Pix-MC)

See Table B.21 for a description of the parameters of the experiment we ran in the Pix-MC Environment. The experiments are the Cartesian product of the table.

Table B.21: Pix-MC parameters

	Parameters
γ	.97
N	512
T	500
$\epsilon-$ Greedy, (π_b, π_e)	$\{(.25, 0), (.1, 0)$ $(.25, .1)\}$
Stochastic Env	False
Stochastic Rew	False
Sparse Rew	False
Seed	{10 of random$(0 : 2^{16})$}
ModelType	{Tabular, NN}
Regress π_b	False

Enduro

See Table B.22 for a description of the parameters of the experiment we ran in the Enduro Environment. The experiments are the Cartesian product of the table.

Table B.22: Enduro parameters

	Parameters
γ	.9999
N	512
T	500
$\epsilon-$ Greedy, (π_b, π_e)	$\{(.25, 0), (.1, 0)$ $(.25, .1)\}$
Stochastic Env	False
Stochastic Rew	False
Sparse Rew	False
Seed	{10 of random$(0 : 2^{16})$}
ModelType	{Tabular, NN}
Regress π_b	False

Representation and Function Class

For the simpler environments, we use a tabular representation for all the methods. AM amounts to solving for the transition dynamics, rewards, terminal state, etc. through maximum likelihood. FQE, Retrace(λ), $Q^{\pi}(\lambda)$, and Tree-Backup are all implemented through dynamics programming with Q tables. MRDR and Q-Reg used the Sherman Morrison Sherman and Morrison (1950) method to solve the weighted-least square problem, using a basis which spans a table.

In the cases where we needed function approximation, we did not directly fit the dynamics for AM; instead, we fit on the difference in states $T(x' - x|x, a)$, which is common practice.

For the MC environment, we ran experiments with both a linear and NN function class. In both cases, the representation of the state was not changed and remained [position, velocity]. The NN architecture was dense with [16,8,4,2] as the layers. The layers had relu activations (except the last, with a linear activation) and were all initialized with truncated normal centered at 0 with a standard deviation of 0.1.

For the pixel-based environments (MC, Enduro), we use a convolutional NN. The architechure is a layer of size 8 with filter (7,7) and stride 3, followed by maxpooling and a layer of size 16 with filter (3,3) and stride 1, followed by max pooling, flattening and a dense layer of size 256. The final layer is a dense layer with the size of the action space, with a linear activation. The layers had elu activations and were all initialized with truncated normal centered at 0 with a standard deviation of 0.1. The layers also have kernel L2 regularizers with weight 1e-6.

When using NNs for the IH method, we used the radial-basis function and a shallow dense network for the kernel and density estimate respectively.

Choice of hyperparameters

Many methods require selection of convergence criteria, regularization parameters, batch sizes, and a whole host of other hyperparameters. Often there is a trade-off between computational cost and the accuracy of the method. Hyperparameter search is not feasible in OPE since there is no proper validation (like game score in learning). See Table B.23 for a list of hyperparameters that were chosen for the experiments.

Table B.23: Hyperparameters for each model by Environment

Method	Parameter	Graph	TMC	MC	Pix-MC	Enduro	POMDP	GW	Pix-GW
AM	Max Traj Len	T	T	50	50	-	T	T	T
	NN Fit Epochs	-	-	100	100	-	-	-	100
	NN Batchsize	-	-	32	32	-	-	-	25
	NN Train size	-	-	.8	.8	-	-	-	.8
	NN Val size	-	-	.2	.2	-	-	-	.2
	NN Stop delta	-	-	1e-4	1e-4	-	-	-	1e-4
Q-Reg	Omega regul.	1	1	-	-	-	1	1	-
	NN Fit Epochs	-	-	80	80	80	-	-	80
	NN Batchsize	-	-	32	32	32	-	-	32
	NN Train size	-	-	.8	.8	.8	-	-	.8
	NN Val size	-	-	.2	.2	.2	-	-	.2
	NN Stop delta	-	-	1e-4	1e-4	1e-4	-	-	1e-4
FQE	Convergence ϵ	1e-5	1e-5	1e-4	1e-4	1e-4	1e-5	4e-4	1e-4
	Max Iter	-	-	160	160	600	-	50	80
	NN Batchsize	-	-	32	32	32	-	-	32
	Optimizer Clipnorm	-	-	1.	1.	1.	-	-	1.
IH	Quad. prog. regular.	1e-3	1e-3	-	-	-	1e-3	1e-3	-
	NN Fit Epochs	-	-	10001	10001	10001	-	-	1001
	NN Batchsize	-	-	1024	128	128	-	-	128
MRDR	Omega regul.	1	1	-	-	-	1	1	-
	NN Fit Epochs	-	-	80	80	80	-	-	80
	NN Batchsize	-	-	1024	1024	1024	-	-	32
	NN Train size	-	-	.8	.8	.8	-	-	.8
	NN Val size	-	-	.2	.2	.2	-	-	.2
	NN Stop delta	-	-	1e-4	1e-4	1e-4	-	-	1e-4
$R(\lambda)$	λ	.9	.9	.9	-	-	.9	.9	.9
	Convergence ϵ	1e-3	2e-3	1e-3	-	-	1e-3	2e-3	1e-3
	Max Iter	500	500	-	-	-	500	50	-
	NN Fit Epochs	-	-	80	-	-	-	-	80
	NN Batchsize	-	-	4	-	-	-	-	4
	NN Train Size	-	-	.03	-	-	-	-	.03
	NN ClipNorm	-	-	1.	-	-	-	-	1.
$Q^{\pi}(\lambda)$	λ	.9	.9	.9	-	-	.9	.9	.9
	Convergence ϵ	1e-3	2e-3	1e-3	-	-	1e-3	2e-3	1e-3
	Max Iter	500	500	-	-	-	500	50	-
	NN Fit Epochs	-	-	80	-	-	-	-	80
	NN Batchsize	-	-	4	-	-	-	-	4
	NN Train Size	-	-	.03	-	-	-	-	.03
	NN ClipNorm	-	-	1.	-	-	-	-	1.
Tree	λ	.9	.9	.9	-	-	.9	.9	.9
	Convergence ϵ	1e-3	2e-3	1e-3	-	-	1e-3	2e-3	1e-3
	Max Iter	500	500	-	-	-	500	50	-
	NN Fit Epochs	-	-	80	-	-	-	-	80
	NN Batchsize	-	-	4	-	-	-	-	4
	NN Train Size	-	-	.03	-	-	-	-	.03
	NN ClipNorm	-	-	1.	-	-	-	-	1.

B.7 Additional Supporting Figures for Chapter 3

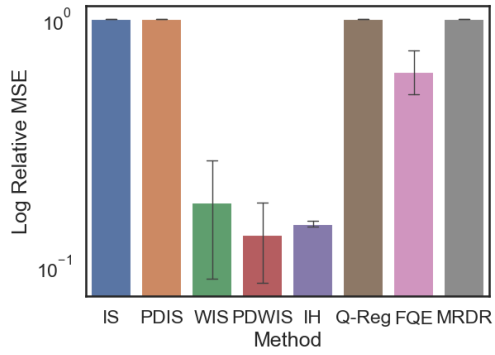


Figure B.7: Enduro DM vs IPS. π_b is a policy that deviates uniformly from a trained policy 25% of the time, π_e is a policy trained with DDQN. *IH has relatively low error mainly due to tracking the simple average, since the kernel function did not learn useful density ratio. The computational time required to calculate the multi-step rollouts of AM, Retrace(λ), $Q^\pi(\lambda)$, Tree-Backup(λ) exceeded our compute budget and were thus excluded.*

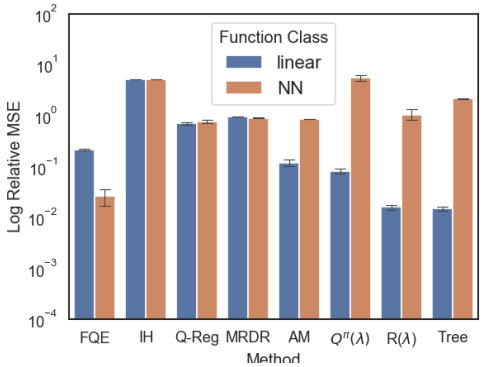


Figure B.8: MC comparison. $N = 256$. π_b is a uniform random policy, π_e is a policy trained with DDQN

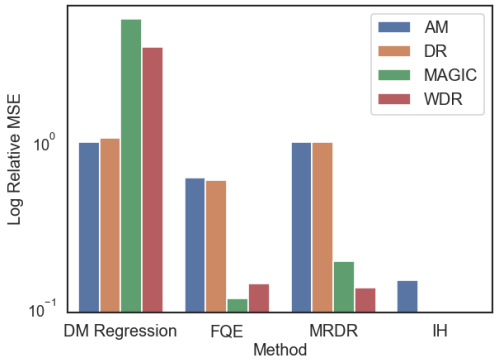


Figure B.9: Enduro DM vs HM. π_b is a policy that deviates uniformly from a trained policy 25% of the time, π_e is a policy trained with DDQN.

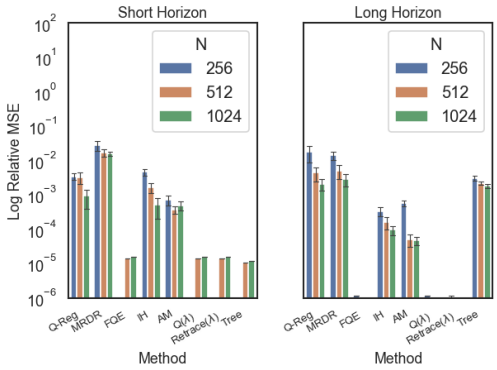


Figure B.10: Comparison of Direct methods' performance across horizon and number of trajectories in the Toy-Graph environment. Small policy mismatch under a deterministic environment.

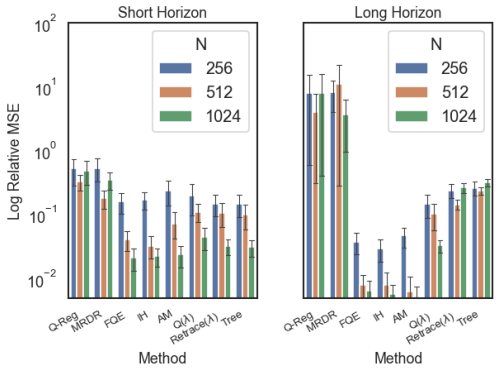


Figure B.11: *(Graph domain) Comparing DMs across horizon length and number of trajectories. Large policy mismatch and a stochastic environment setting.*

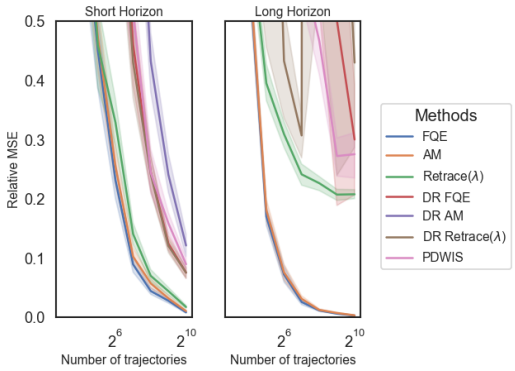


Figure B.12: Comparing DM to DR in a stochastic environment with large policy mismatch. (Graph)

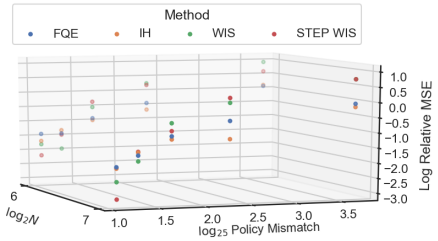


Figure B.13: Comparison between FQE, IH and WIS in a low data regime. For low policy mismatch, IPS is competitive to DM in low data, but as the policy mismatch grows, the top DM outperform. Experiments ran in the Gridworld Environment.

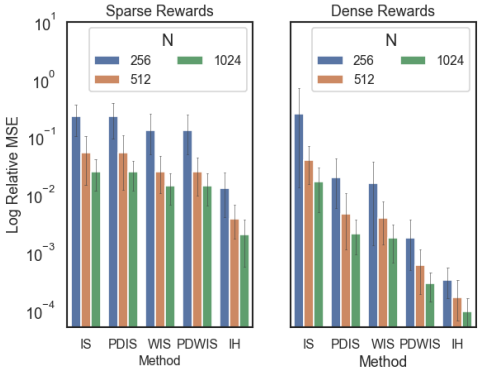


Figure B.14: Comparison between IPS methods and IH with dense vs sparse rewards. Per-Decision IPS methods see substantial improvement when the rewards are dense. Experiments ran in the Toy-Graph environment with $\pi(a=0)=.6, \pi_e(a=0)=.8$ See Tables B.27, B.28, B.29, B.24, B.25, B.26

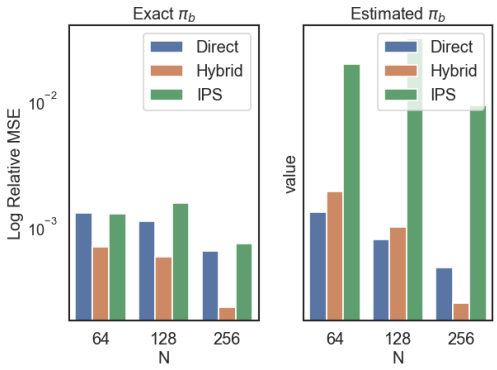


Figure B.15: Exact vs Estimated π_b. Exact π_b = .2−Greedy(optimal), π_e = .1−Greedy(optimal). Min error per class. (Pixel Gridworld, deterministic)

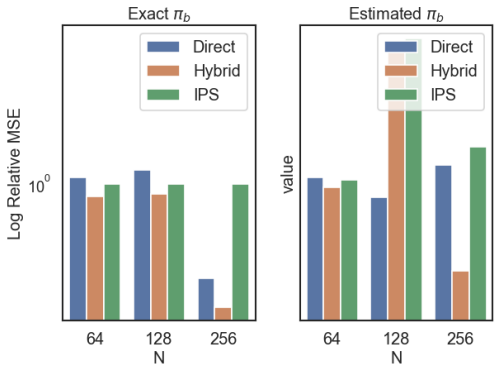


Figure B.16: Exact vs Estimated π_b. Exact π_b =uniform, π_e = .1−Greedy(optimal). Min error per class. (Pixel Gridworld, deterministic)

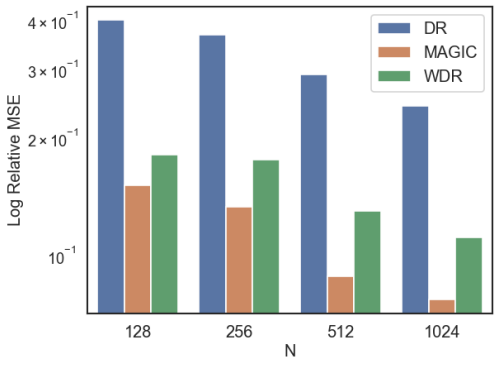


Figure B.17: Hybrid Method comparison. $\pi_b(a = 0) = .2, \pi_e(a = 0) = .8$. Min error per class. (Graph-MC)

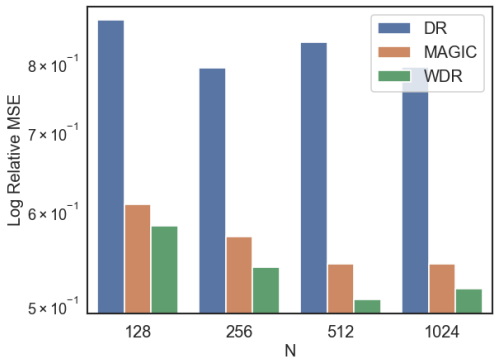


Figure B.18: Hybrid Method comparison. $\pi_b(a=0) = .8, \pi_e(a=0) = .2$. Min error per class. (Graph-MC)

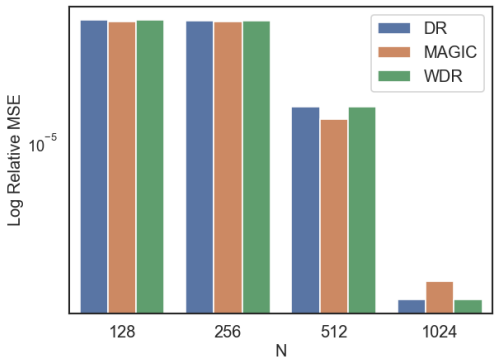


Figure B.19: Hybrid Method comparison. $\pi_b(a=0) = .6, \pi_e(a=0) = .6$. Min error per class. (Graph-MC)

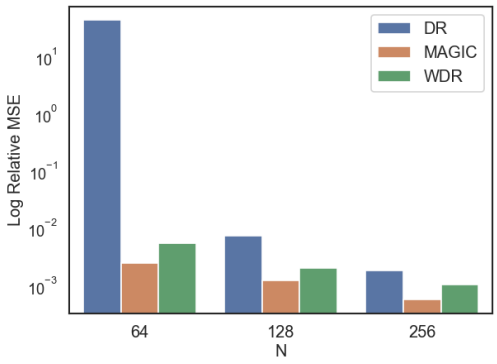


Figure B.20: Hybrid Method comparison. Exact $\pi_b = .2-$Greedy(optimal), $\pi_e = .1-$Greedy(optimal). Min error per class. (Pixel Gridworld)

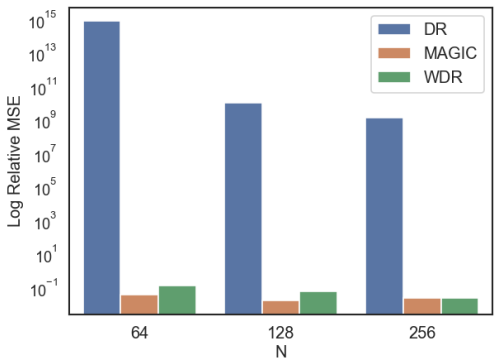


Figure B.21: Hybrid Method comparison. $\pi_b = .8-$Greedy(optimal), $\pi_e = .1-$Greedy(optimal). Min error per class. (Pixel Gridworld)

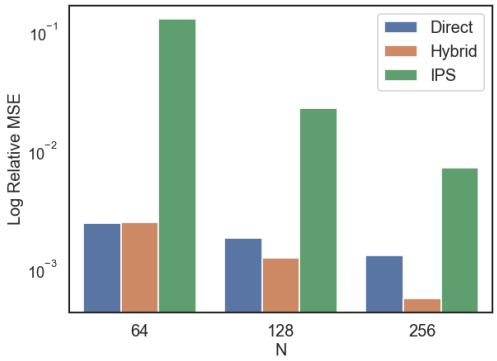


Figure B.22: Class comparison with unknown π_b. At first, HM underperform DM because π_b is more difficult to calculate leading to imprecise importance sampling estimates. Exact $\pi_b = .2-$Greedy(optimal), $\pi_e = .1-$Greedy(optimal). Min error per class. (Pixel Gridworld, stochastic env with .2 slippage)

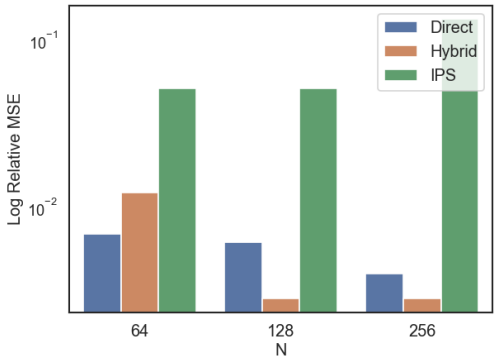


Figure B.23: Class comparison with unknown π_b. At first, HM underperform DM because π_b is more difficult to calculate leading to imprecise importance sampling estimates. Exact $\pi_b = .6-$Greedy(optimal), $\pi_e = .1-$Greedy(optimal). Min error per class. (Pixel Gridworld, stochastic env with .2 slippage)

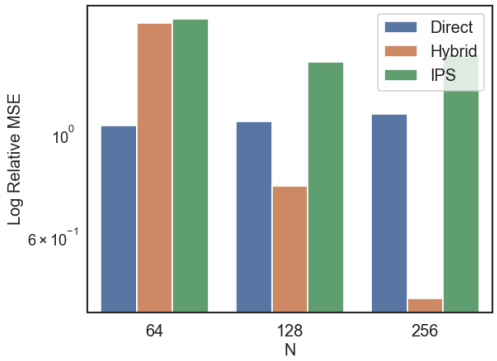


Figure B.24: Class comparison with unknown π_b. At first, HM underperform DM because π_b is more difficult to calculate leading to imprecise importance sampling estimates. Exact π_b =uniform, π_e = .1−Greedy(optimal). Min error per class. (Pixel Gridworld, stochastic env with .2 slippage)

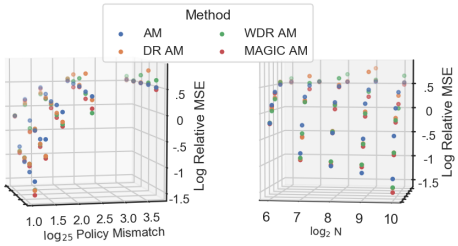


Figure B.25: AM Direct vs Hybrid comparison for AM. (Gridworld)

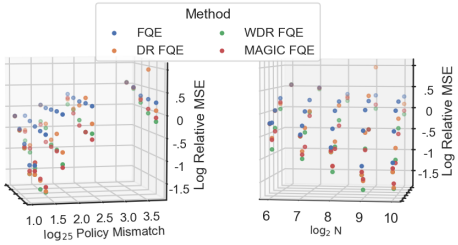


Figure B.26: FQE Direct vs Hybrid comparison. (Gridworld)

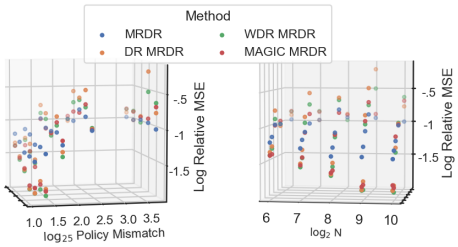


Figure B.27: MRDR Direct vs Hybrid comparison. (Gridworld)

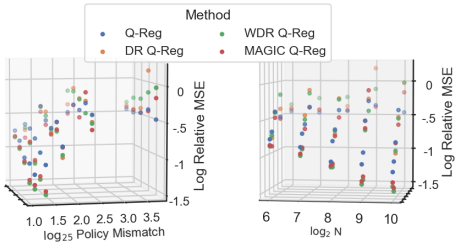


Figure B.28: Q-Reg Direct vs Hybrid comparison. (Gridworld)

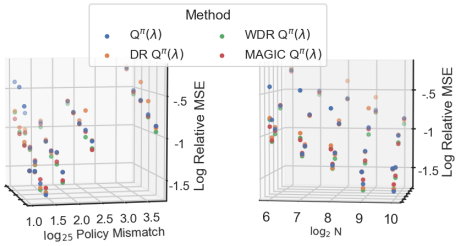


Figure B.29: $Q^{\pi}(\lambda)$ Direct vs Hybrid comparison. (Gridworld)

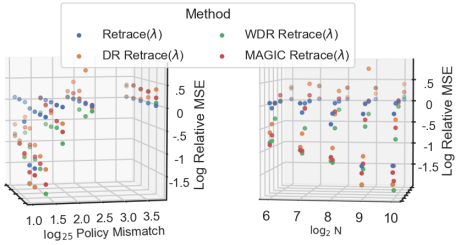


Figure B.30: Retrace(λ) Direct vs Hybrid comparison. (Gridworld)

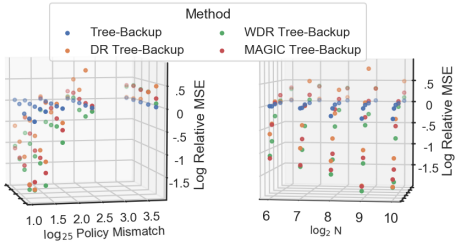


Figure B.31: Tree-Backup Direct vs Hybrid comparison. (Gridworld)

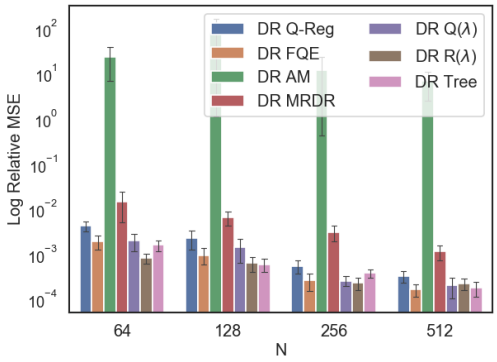


Figure B.32: DR comparison with $\pi_b = .2-$Greedy(optimal), $\pi_e = 1.-$Greedy(optimal). (Pixel Gridworld)

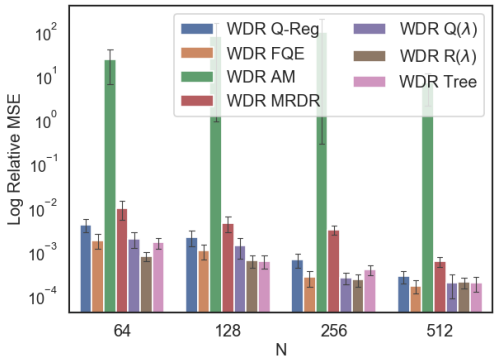


Figure B.33: WDR comparison with $\pi_b = .2-$Greedy(optimal), $\pi_e = 1.-$Greedy(optimal). (Pixel Gridworld)

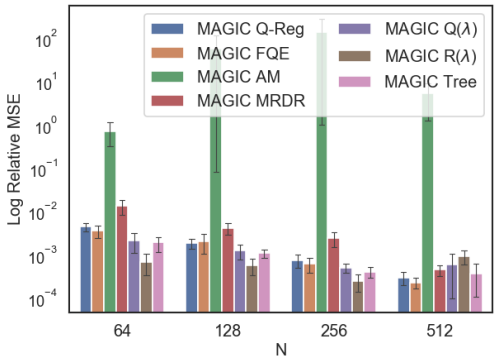


Figure B.34: MAGIC comparison with $\pi_b = .2-$Greedy(optimal), $\pi_e = 1.-$Greedy(optimal). (Pixel Gridworld)

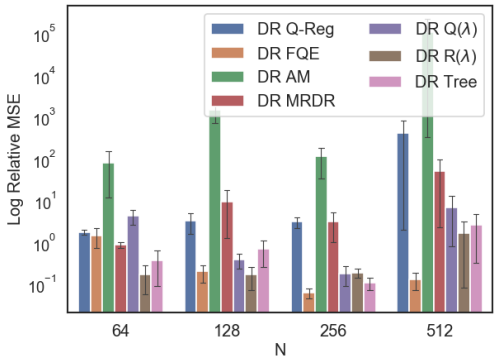


Figure B.35: DR comparison with $\pi_b = .8-$Greedy(optimal), $\pi_e = 1.-$Greedy(optimal). (Pixel Gridworld)

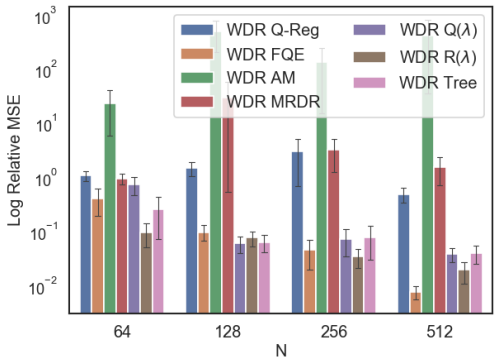


Figure B.36: WDR comparison with $\pi_b = .8-$Greedy(optimal), $\pi_e = 1.-$Greedy(optimal). (Pixel Gridworld)

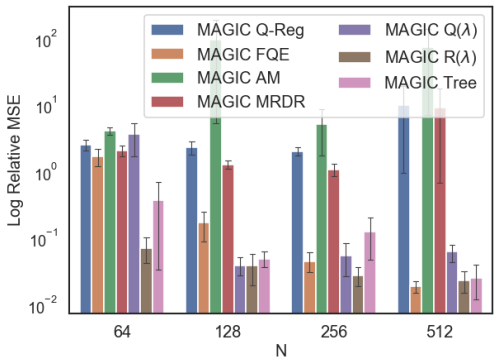


Figure B.37: MAGIC comparison with $\pi_b = .8-$Greedy(optimal), $\pi_e = 1.-$Greedy(optimal). (Pixel Gridworld)

B.8 Additional Supporting Tables to Chapter 3

Table B.24: Graph, relative MSE. $T = 16, N = 256, \pi_b(a = 0) = 0.2, \pi_e(a = 0) = 0.8$. Sparse rewards.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	2.4E-3	2.4E1	7.3E0	3.2E-2
Q-Reg	9.3E-1	1.2E1	1.2E1	8.4E-1
MRDR	8.6E-1	4.6E0	1.5E2	1.5E2
FQE	**5.1E-5**	5.1E-5	5.1E-5	5.1E-5
R(λ)	1.0E0	9.3E-1	2.0E0	2.2E0
Q$^\pi$(λ)	5.1E-5	5.1E-5	5.1E-5	**5.0E-5**
Tree	1.0E0	9.3E-1	2.0E0	2.2E0
IH	1.5E-1	-	-	-

	IPS	
	Standard	Per-Decision
IS	**9.3E-1**	9.3E-1
WIS	2.0E0	2.0E0
NAIVE	4.0E0	-

Table B.25: Graph, relative MSE. $T = 16, N = 512, \pi_b(a = 0) = 0.2, \pi_e(a = 0) = 0.8$. Sparse rewards.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	1.6E-3	1.8E1	4.6E0	2.6E-2
Q-Reg	1.7E1	9.2E2	6.1E2	2.4E1
MRDR	9.5E0	9.6E2	3.7E2	1.3E2
FQE	**5.0E-6**	5.0E-6	5.0E-6	**5.0E-6**
R(λ)	1.0E0	1.6E1	1.9E0	1.6E0
Q$^\pi$(λ)	5.0E-6	1.1E-5	5.0E-6	5.0E-6
Tree	1.0E0	1.6E1	1.9E0	1.6E0
IH	8.4E-2	-	-	-

	IPS	
	Standard	Per-Decision
IS	1.6E1	1.6E1
WIS	1.9E0	**1.9E0**
NAIVE	3.9E0	-

Table B.26: Graph, relative MSE. $T = 16, N = 1024, \pi_b(a = 0) = 0.2, \pi_e(a = 0) = 0.8$. Sparse rewards.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	1.8E-3	2.3E3	2.7E0	3.2E-3
Q-Reg	1.2E3	2.2E3	2.5E1	1.3E3
MRDR	1.8E4	2.5E4	1.4E2	9.6E2
FQE	**2.4E-5**	2.4E-5	2.4E-5	2.4E-5
R(λ)	1.0E0	1.1E3	2.5E0	1.0E0
Q$^\pi$(λ)	2.4E-5	**2.3E-5**	2.4E-5	2.4E-5
Tree	1.0E0	1.1E3	2.5E0	1.0E0
IH	2.6E-2	-	-	-

	IPS	
	Standard	Per-Decision
IS	1.1E3	1.1E3
WIS	2.5E0	**2.5E0**
NAIVE	3.9E0	-

Table B.27: Graph, relative MSE. $T = 16, N = 256, \pi_b(a = 0) = 0.6, \pi_e(a = 0) = 0.8$. Dense rewards.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	5.8E-4	1.5E-3	1.6E-3	6.5E-4
Q-Reg	1.8E-2	2.3E-3	9.4E-4	9.9E-4
MRDR	1.4E-2	3.7E-2	2.5E-2	1.7E-2
FQE	1.0E-6	1.0E-6	1.0E-6	1.0E-6
R(λ)	**1.0E-6**	1.0E-6	**1.0E-6**	1.0E-6
Q$^\pi$(λ)	1.0E-6	1.0E-6	1.0E-6	1.0E-6
Tree	3.0E-3	2.6E-4	2.4E-4	2.4E-4
IH	3.4E-4	-	-	-

	IPS	
	Standard	Per-Decision
IS	2.6E-1	2.0E-2
WIS	1.6E-2	**1.8E-3**
NAIVE	4.5E-1	-

Table B.28: Graph, relative MSE. $T = 16, N = 512, \pi_b(a = 0) = 0.6, \pi_e(a = 0) = 0.8$. Dense rewards.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	5.0E-5	3.3E-3	3.3E-3	1.5E-3
Q-Reg	4.5E-3	7.7E-5	3.3E-5	4.5E-5
MRDR	5.0E-3	2.6E-3	2.7E-3	5.7E-3
FQE	7.7E-7	7.7E-7	7.7E-7	7.7E-7
R(λ)	8.1E-7	8.0E-7	7.9E-7	8.1E-7
Q$^\pi$(λ)	**7.6E-7**	7.7E-7	7.7E-7	**7.6E-7**
Tree	2.2E-3	1.2E-4	1.1E-4	1.1E-4
IH	1.7E-4	-	-	-

	IPS	
	Standard	Per-Decision
IS	4.0E-2	4.7E-3
WIS	4.0E-3	**6.0E-4**
NAIVE	4.5E-1	-

Table B.29: Graph, relative MSE. $T = 16, N = 1024, \pi_b(a = 0) = 0.6, \pi_e(a = 0) = 0.8$. Dense rewards.

	DM	Hybrid		
	Direct	DR	WDR	MAGIC
AM	4.8E-5	4.7E-4	4.8E-4	7.2E-5
Q-Reg	2.1E-3	1.4E-5	1.0E-5	2.9E-5
MRDR	2.9E-3	5.0E-4	4.6E-4	9.6E-4
FQE	2.7E-7	2.7E-7	2.7E-7	2.7E-7
R(λ)	2.9E-7	2.8E-7	2.8E-7	2.9E-7
Q$^\pi$(λ)	**2.7E-7**	2.7E-7	2.7E-7	**2.7E-7**
Tree	1.8E-3	7.7E-5	5.8E-5	5.8E-5
IH	9.6E-5	-	-	-

	IPS	
	Standard	Per-Decision
IS	1.7E-2	2.1E-3
WIS	1.8E-3	**2.9E-4**
NAIVE	4.4E-1	-

Appendix C

APPENDIX TO CHAPTER 4

C.1 Detailed Theoretical Analysis and Proofs

Proof of lemma 4.5.1

Lemma Statement. *(Lemma 4.5.1) For a fixed x, define* $\pi([x,a]) \triangleq \varphi(a)$. *If* φ *is non-negative and H-smooth w.r.t. a., then:*

$$\forall a, a' : \ \left(\varphi(a) - \varphi(a')\right)^2 \leqslant 6H\left(\varphi(a) + \varphi(a')\right)\left\|a - a'\right\|^2.$$

The proof of Lemma 4.5.1 rests on 2 properties of H-smooth functions (differentiable) in $\mathbb{R}^1$, as stated below

Lemma C.1.1 (Self-bounding property of Lipschitz-smooth functions)**.** *Let* $\phi : \mathbb{R} \to \mathbb{R}$ *be an H-smooth non-negative function. Then for all* $a \in \mathbb{R}$*:* $|\nabla\phi(a)| \leqslant \sqrt{4H\phi(a)}$

Proof. By mean value theorem, for any a, a' we have $\exists\ \eta \in (a, a')$ (or (a', a)) such that $\phi(a') = \phi(a) + \nabla\phi(\eta)(a' - a)$. Since ϕ is non-negative,

$$\begin{aligned} 0 \leqslant \phi(a') &= \phi(a) + \nabla\phi(a)(a' - a) \\ &\quad + (\nabla\phi(\eta) - \nabla\phi(a))(a' - a) \\ &\leqslant \phi(a) + \nabla\phi(a)(a' - a) + H|\eta - a||a' - a| \\ &\leqslant \phi(a) + \nabla\phi(a)(a' - a) + H|a' - a|^2 \end{aligned}$$

Choosing $a' = a - \frac{\nabla\phi(a)}{2H}$ proves the lemma. □

Lemma C.1.2 (1-d Case (Srebro et al., 2010))**.** *Let* $\phi : \mathbb{R} \to \mathbb{R}$ *be an H-smooth non-negative function. Then for all* $a, a' \in \mathbb{R}$*:*

$$\left(\phi(a) - \phi(a')\right)^2 \leqslant 6H\left(\phi(a) + \phi(a')\right)\left(a - a'\right)^2$$

Proof. As before, $\exists\eta \in (a, a')$ such that $\phi(a') - \phi(a) = \nabla\phi(\eta)(a' - a)$. By assumption of ϕ, we have $|\nabla\phi(\eta) - \nabla\phi(a)| \leqslant H|\eta - a| \leqslant H|a' - a|$. Thus we have:

$$|\nabla\phi(\eta)| \leqslant |\nabla\phi(a)| + H|a - a'| \tag{C.1}$$

Consider two cases:

Case 1: If $|a-a'| \leqslant \frac{|\nabla\phi(a)|}{5H}$, then by equation C.1 we have $|\nabla\phi(\eta)| \leqslant 6/5|\nabla\phi(a)|$. Thus

$$\begin{aligned}\left(\phi(a)-\phi(a')\right)^2 &= \left(\nabla\phi(\eta)\right)^2\left(a-a'\right)^2\\ &\leqslant \frac{36}{25}\left(\nabla\phi(a)\right)^2\left(a-a'\right)^2\\ &\leqslant \frac{144}{25}H\phi(a)\left(a-a'\right)^2\end{aligned}$$

by lemma C.1.1. Therefore:

$$\left(\phi(a)-\phi(a')\right)^2 \leqslant 6H\phi(a)\left(a-a'\right)^2 \leqslant 6H\left(\phi(a)+\phi(a')\right)\left(a-a'\right)^2$$

Case 2: If $|a-a'| > \frac{|\nabla\phi(a)|}{5H}$, then equation C.1 gives $|\nabla\phi(\eta)| \leqslant 6H|a-a'|$. Once again

$$\begin{aligned}\left(\phi(a)-\phi(a')\right)^2 &= \left(\phi(a)-\phi(a')\right)\nabla\phi(\eta)\left(a-a'\right)\\ &\leqslant |\left(\phi(a)-\phi(a')\right)||\nabla\phi(\eta)||\left(a-a'\right)|\\ &\leqslant |\left(\phi(a)-\phi(a')\right)|\left(6H\left(a-a'\right)^2\right)\\ &\leqslant 6H\left(\phi(a)+\phi(a')\right)\left(a-a'\right)^2\end{aligned}$$

□

Proof of Lemma 4.5.1. The extension to the multi-dimensional case is straightforward. For any $a, a' \in \mathbb{R}^k$, consider the function $\phi : \mathbb{R} \to \mathbb{R}$ such that $\phi(t) = \varphi((1-t)a + ta')$, then ϕ is a differentiable, non-negative function and $\nabla_t(\phi(t)) = \langle\nabla\varphi(a+t(a'-a)), a'-a\rangle$. Thus:

$$\begin{aligned}|\phi'(t_1)-\phi'(t_2)| &= |\langle\nabla\varphi(a+t_1(a'-a))-\\ &\qquad\qquad \nabla\varphi(a+t_2(a'-a)), a'-a\rangle|\\ &\leqslant \left\|\nabla\varphi(a+t_1(a'-a))-\nabla\varphi(a+t_2(a'-a))\right\|_* \left\|a'-a\right\|\\ &\leqslant H|t_1-t_2|\left\|a-a'\right\|^2\end{aligned}$$

Therefore ϕ is an $H\left\|a-a'\right\|^2$-smooth function in $\mathbb{R}$. Apply lemma C.1.2 to ϕ, we have:

$$\left(\phi(1)-\phi(0)\right)^2 \leqslant 6H\left\|a-a'\right\|^2\left(\phi(1)+\phi(0)\right)(1-0)^2$$

which is the same as $(\varphi(a)-\varphi(a'))^2 \leqslant 6H(\varphi(a)+\varphi(a'))\left\|a-a'\right\|^2$ □

Proof of lemma 4.5.2

Lemma Statement. *(Lemma 4.5.2) Given any starting state s_0, sequentially execute π_{det} and π_{sto} to obtain two separate trajectories $\boldsymbol{A} = \{a_t\}_{t=1}^T$ and $\tilde{\boldsymbol{A}} = \{\tilde{a}_t\}_{t=1}^T$ such that $a_t = \pi_{det}(s_t)$ and $\tilde{a}_t = \pi_{sto}(\tilde{s}_t)$, where $s_t = [x_t, a_{t-1}]$ and $\tilde{s}_t = [x_t, \tilde{a}_{t-1}]$. Assuming the policies are stable as per Condition 1, we have $\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] = a_t \ \forall t = 1, \ldots, T$, where the expectation is taken over all random roll-outs of π_{sto}.*

Proof. Given a starting state s_0, we prove by induction that $\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] = a_t$.

It is easily seen that the claim is true for $t = 1$.

Now assuming that $\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_{t-1}] = a_{t-1}$. We have

$$\begin{aligned}
\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] &= \mathbb{E}_{\tilde{\boldsymbol{A}}}[\mathbb{E}[\tilde{a}_t|\tilde{s}_t]] \\
&= \mathbb{E}_{\tilde{\boldsymbol{A}}}[\beta\hat{\pi}(\tilde{s}_t) + (1-\beta)\pi(\tilde{s}_t)] \\
&= \beta\mathbb{E}_{\tilde{\boldsymbol{A}}}[\hat{\pi}(\tilde{s}_t)] + (1-\beta)\mathbb{E}_{\tilde{\boldsymbol{A}}}[\pi(\tilde{s}_t)]
\end{aligned}$$

Thus:

$$\begin{aligned}
\|\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] - a_t\| &= \|\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] - \beta\hat{\pi}(s_t) - (1-\beta)\pi(s_t)\| \\
&= \|\beta\mathbb{E}_{\tilde{\boldsymbol{A}}}[\hat{\pi}(\tilde{s}_t)] + (1-\beta)\mathbb{E}_{\tilde{\boldsymbol{A}}}[\pi(\tilde{s}_t)] \\
&\qquad - \beta\hat{\pi}(s_t) - (1-\beta)\pi(s_t)\| \\
&\leqslant \beta\,\|\mathbb{E}_{\tilde{\boldsymbol{A}}}[\hat{\pi}(\tilde{s}_t)] - \hat{\pi}(s_t)\| \\
&\qquad + (1-\beta)\,\|\mathbb{E}_{\tilde{\boldsymbol{A}}}[\pi(\tilde{s}_t)] - \pi(s_t)\| \\
&\leqslant \beta\,\|\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_{t-1}] - a_{t-1}\| \\
&\qquad + (1-\beta)\,\|\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_{t-1}] - a_{t-1}\| \\
&= 0
\end{aligned}$$

per inductive hypothesis. Therefore we conclude that $\mathbb{E}_{\tilde{\boldsymbol{A}}}[\tilde{a}_t] = a_t \ \forall t = 1, \ldots, T$ □

Proof of theorem 4.5.6 and corollary 4.5.7 - Main policy improvement results

In this section, we provide the proof to theorem 4.5.6 and corollary 4.5.7.

Theorem Statement. *(theorem 4.5.6) Assume ℓ is convex and L-Lipschitz-continuous, and Condition 2 holds. Let $\epsilon = \max_{s\sim d_\pi} \|\hat{\pi}(s) - \pi(s)\|$. Then for $\beta \in (0,1)$:*

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant \frac{\beta\gamma\epsilon L}{(1-\beta)(1-\gamma)} + \beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)).$$

Proof. First let's review the notations: let T be the trajectory horizon. For a policy π in the deterministic policy class Π, given a starting state s_0, we roll out the full trajectory $s_0 \xrightarrow{\pi} s_1 \xrightarrow{\pi} \ldots \xrightarrow{\pi} s_T$, where $s_t = [x_t, \pi(s_{t-1})]$, with x_t encodes the featurized input at current time t, and $\pi(s_{t-1})$ encodes the dependency on previous predictions. Let $\ell(\pi(s))$ be the loss of taking action $\pi(s)$ at state s, we can define the trajectory loss of policy π from starting state s_0 as

$$\ell(\pi|s_0) = \frac{1}{T}\sum_{t=1}^{T} \ell(\pi(s_t))$$

For a starting state distribution μ, we define policy loss of π as the expected loss along trajectories induced by π: $\ell_\pi(\pi) = \mathbb{E}_{s_0\sim\mu}[\ell(\pi|s_0)]$. Policy loss $\ell_\pi(\pi)$ can be understood as

$$\ell_\pi(\pi) = \int_{s_0\sim\mu} \mathop{\mathbb{E}}_{x_t\sim\mathcal{X}} \frac{1}{T}\left[\sum_{t=1}^{T} \ell(\pi(s_t))\right] d\mu(s_0)$$

To prove policy improvement, we skip the subscript of algorithm 4 to consider general policy update rule within each iteration:

$$\pi' = \pi_{new} = \beta\hat{\pi} + (1-\beta)\pi \tag{C.2}$$

where $\pi = \pi_{old}$ is the current policy (combined up until the previous iteration), $\hat{\pi}$ is the trained model from calling the base regression routine $\texttt{Train}(\mathbf{S}, \widehat{\mathbf{A}}|h)$. Learning rate (step-size) β may be adaptively chosen in each iteration. Recall that this update rule reflects deterministic interpolation of two policies.

We are interested in quantifying the policy improvement when updating π to π'. Specifically, we want to bound

$$\Gamma = \ell_{\pi'}(\pi') - \ell_\pi(\pi)$$

where $\ell_\pi(\pi)$ (respectively $\ell_{\pi'}(\pi')$) denotes the trajectory loss of π (respectively π') on the state distribution induced by π (resp. π')

We will bound the loss difference of old and new policies conditioned on a common starting state s_0. Based on update rule (C.2), consider rolling out π' and π from the same starting state s_0 to obtain two separate sequences $\pi' \longmapsto \{s_0 \to s'_1 \ldots \to s'_T\}$ and $\pi \longmapsto \{s_0 \to s_1 \ldots \to s_T\}$ corresponding to the same stream of inputs

$x_1, \ldots, x_T$.

$$\begin{aligned}\Gamma(s_0) &= \frac{1}{T}\sum_{t=1}^{T} \ell(\pi'(s'_t)) - \ell(\pi(s_t)) \\ &= \frac{1}{T}\sum_{t=1}^{T} \ell(\pi'(s'_t)) - \ell(\pi'(s_t)) + \ell(\pi'(s_t)) - \ell(\pi(s_t)) \qquad \text{(C.3)}\end{aligned}$$

Assume convexity of ℓ (e.g. sum of square losses):

$$\begin{aligned}\ell(\pi'(s_t)) &= \ell(\beta\hat{\pi}(s_t) + (1-\beta)\pi(s_t)) \\ &\leqslant \beta\ell(\hat{\pi}(s_t)) + (1-\beta)\ell(\pi(s_t))\end{aligned}$$

Thus we can begin to bound individual components of $\Gamma(s_0)$ as

$$\begin{aligned}\ell(\pi'(s'_t)) - \ell(\pi(s_t)) \leqslant \ell(\pi'(s'_t))) - \ell(\pi'(s_t)) \\ + \beta\left[\ell(\hat{\pi}(s_t)) - \ell(\pi(s_t))\right]\end{aligned}$$

Since ℓ is L-Lipschitz continuous, we have

$$\begin{aligned}\ell(\pi'(s'_t)) - \ell(\pi'(s_t)) &\leqslant L\left\|\pi'(s'_t) - \pi'(s_t)\right\| \\ &\leqslant L\gamma\left\|s'_t - s_t\right\| \qquad \text{(C.4)}\end{aligned}$$

where (C.4) is due to the smoothness condition [2] of policy class Π. Given a policy class Π with $\gamma < 1$, the following claim can be proved by induction:
Claim: $\|s'_t - s_t\| \leqslant \frac{\beta\epsilon}{(1-\beta)(1-\gamma)}$

Proof. For the base case, given the same starting state s_0, we have $s'_1 = [x_1, \pi'(s_0)]$ and $s_1 = [x_1, \pi(s_0)]$. Thus

$$\begin{aligned}\left\|s'_1 - s_1\right\| = \left\|\pi'(s_0) - \pi(s_0)\right\| &= \left\|\beta\hat{\pi}(s_0) + (1-\beta)\pi(s_0) - \pi(s_0)\right\| \\ = \beta\left\|\hat{\pi}(s_0) - \pi(s_0)\right\| \leqslant \beta\epsilon &\leqslant \frac{\beta\epsilon}{(1-\beta)(1-\gamma)}\end{aligned}$$

In the inductive case, assume we have $\left\|s'_{t-1} - s_{t-1}\right\| \leqslant \frac{\beta\epsilon}{(1-\beta)(1-\gamma)}$. Then similar to

before, the definition of s'_t and s_t leads to

$$\begin{aligned}
\left\|s'_t - s_t\right\| &= \left\|\left[x_t, \pi'(s'_{t-1})\right] - \left[x_t, \pi(s_{t-1})\right]\right\| \\
&= \left\|\pi'(s'_{t-1}) - \pi(s_{t-1})\right\| \\
&\leqslant \left\|\pi'(s'_{t-1}) - \pi'(s_{t-1})\right\| + \left\|\pi'(s_{t-1}) - \pi(s_{t-1})\right\| \\
&\leqslant \gamma \left\|s'_{t-1} - s_{t-1}\right\| + \beta \left\|\hat{\pi}(s_{t-1}) - \pi(s_{t-1})\right\| \\
&\leqslant \gamma \frac{\beta\epsilon}{(1-\beta)(1-\gamma)} + \beta\epsilon \\
&\leqslant \frac{\beta\epsilon}{(1-\beta)(1-\gamma)}
\end{aligned}$$

□

Applying the claim to equation (C.4), we have

$$\ell(\pi'(s'_t)) - \ell(\pi'(s_t)) \leqslant \frac{\beta\gamma\epsilon L}{(1-\beta)(1-\gamma)}$$

which leads to

$$\begin{aligned}
\ell(\pi'(s'_t) - \ell(\pi(s_t))) \leqslant \frac{\beta\gamma\epsilon L}{(1-\beta)(1-\gamma)} \\
+ \beta(\ell(\hat{\pi}(s_t)) - \ell(\pi(s_t)))
\end{aligned} \tag{C.5}$$

Integrating (C.5) over the starting state $s_0 \sim \mu$ and input trajectories $\{x_t\}_{t=1}^T$, we arrive at the policy improvement bound:

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant \frac{\beta\gamma\epsilon L}{(1-\beta)(1-\gamma)} + \beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi))$$

where $\ell_\pi(\hat{\pi})$ is the expected loss of the trained policy $\hat{\pi}$ on the state distribution induced by policy π (reduction term, analogous to policy advantage in the traditional MDP terminologies (Kakade and Langford, 2002)) □

This means in the worst case, as we choose $\beta \to 0$, we have $[\ell_{\pi'}(\pi') - \ell_\pi(\pi)] \to 0$, meaning the new policy does not degrade much for a small choice of β. However if $\ell_\pi(\hat{\pi}) - \ell_\pi(\pi) \ll 0$, we can choose β to enforce monotonic improvement of the policy by adaptively choosing β that minimizes the right-hand side. In particular, let the reduction term be $\Delta = \ell_\pi(\pi) - \ell_\pi(\hat{\pi}) > 0$ and let $\delta = \frac{\gamma\epsilon L}{1-\gamma}$, then for $\beta = \frac{\Delta-\delta}{2\Delta}$we have the following monotonic policy improvement:

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant -\frac{(\Delta-\delta)^2}{2(\Delta+\delta)}$$

Proof of theorem 4.5.5 - T-dependent improvement

Theorem Statement. *(theorem 4.5.5) Assume ℓ is convex and L-Lipschitz, and Condition 1 holds. Let $\epsilon = \max_{s \sim d_\pi} \|\hat{\pi}(s) - \pi(s)\|$. Then:*

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant \beta\epsilon LT + \beta\left(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)\right).$$

In particular, choosing $\beta \in (0, 1/T)$ yields:

$$\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant \epsilon L + \beta\left(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)\right).$$

Proof. The proof of theorem 4.5.5 largely follows the structute of theorem 4.5.6, except that we are using the slighty weaker Condition 1 which leads to weaker bound on the policy improvement that depends on the trajectory horizon T. For any state s_0 taken from the starting state distribution μ, sequentially roll-out policies π' and π to receive two separate trajectories $\pi' : s_0 \to s'_1 \to \ldots \to s'_T$ and $\pi' : s_0 \to s_1 \to \ldots \to s_T$. Consider a pair of states $s'_t = [x_t, \pi'(s'_{t-1})]$ and $s_t = [x_t, \pi(s_{t-1})]$ corresponding to the same input feature x_t, as before we can decompose $\ell(\pi'(s'_t)) - \ell(\pi(s_t)) = \ell(\pi'(s'_t)) - \ell(\pi'(s_t)) + \ell(\pi'(s_t)) - \ell(\pi(s_t)) \leqslant L\|\pi'(s'_t) - \pi'(s_t)\| + \beta(\ell(\hat{\pi}(s_t)) - \ell(\pi(s_t)))$ due to convexity and L-Lipschitz continuity of ℓ.

Condition 1 further yields: $\ell(\pi'(s'_t)) - \ell(\pi(s_t)) \leqslant L\|s'_t - s_t\| + \beta(\ell(\hat{\pi}(s_t)) - \ell(\pi(s_t)))$. By the construction of the states, note that

$$\begin{aligned}
\|s'_t - s_t\| &= \|\pi'(s'_{t-1}) - \pi(s_{t-1})\| \\
&\leqslant \|\pi'(s'_{t-1}) - \pi'(s_{t-1})\| + \|\pi'(s_{t-1}) - \pi(s_{t-1})\| \\
&\leqslant \|s'_{t-1} - s_{t-1}\| + \beta(\|\hat{\pi}(s_{t-1}) - \pi(s_{t-1})\|) \\
&\leqslant \|s'_{t-1} - s_{t-1}\| + \beta\epsilon
\end{aligned}$$

(by condition 1 and definition of ϵ).

From here, one can use this recursive relation to easily show that $\|s'_t - s_t\| \leqslant \beta\epsilon t$ for all $t \in [1, T]$.

Averaging over the T time steps and integrating over the starting state distribution, we have:

$$\begin{aligned}
\ell_{\pi'}(\pi') - \ell_\pi(\pi) &\leqslant \beta\epsilon L(T+1)/2 + \beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi)) \\
&\leqslant \beta\epsilon LT + \beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi))
\end{aligned}$$

In particular, $\beta \in (0, 1/T)$ yields $\ell_{\pi'}(\pi') - \ell_\pi(\pi) \leqslant \epsilon L + \beta(\ell_\pi(\hat{\pi}) - \ell_\pi(\pi))$. □

Proof of proposition 4.5.8 - smooth expert proposition

Proposition Statement. *(Proposition 4.5.8) Let ω be the average supervised training error from $\mathcal{F}$, i.e. $\omega = \min_{f\in\mathcal{F}} \mathbb{E}_{x\sim\mathcal{X}}\left[\|f([x,0]) - a^*\|\right]$. Let the rolled-out trajectory of current policy π be $\{a_t\}$. If the average gap between π and π^* is such that $\mathbb{E}_{t\sim Uniform[1:T]}\left[\|a_t^* - a_{t-1}\|\right] \geqslant 3\omega + \eta(1+\lambda)$, then using $\{a_t^*\}$ as feedback will cause the trained policy $\hat{\pi}$ to be non-smooth, i.e.:*

$$\mathbb{E}_{t\sim Uniform[1:T]}\left[\|\hat{a}_t - \hat{a}_{t-1}\|\right] \geqslant \eta,$$

for $\{\hat{a}_t\}$ the rolled-out trajectory of $\hat{\pi}$.

Proof. Recall that Π_λ is formed by regularizing a class of supervised learners $\mathcal{F}$ with the singleton class of smooth function $\mathcal{H} \triangleq \{h(a) = a\}$, via a hyper-parameter λ that controls the trade-off between being close to the two classes.

Minimizing over Π_λ can be seen as a regularized optimization problem:

$$\begin{aligned}
\hat{\pi}(x,a) &= \arg\min_{\pi\in\Pi} \ell(\pi([x,a])) \\
&= \arg\min_{f\in\mathcal{F}, h\in\mathcal{H}} (f(x,a) - a^*)^2 + \lambda(f(x,a) - h(a))^2 \\
&= \arg\min_{f\in\mathcal{F}} (f(x,a) - a^*)^2 + \lambda(f(x,a) - a)^2
\end{aligned} \tag{C.6}$$

where hyper-parameter λ trades-off the distance of $f(x,a)$ relative to a (smoothness) and a^* (imitation accuracy), and $a \in \mathbb{R}^1$.

Such a policy π, at execution time, corresponds to the regularized minimizer of:

$$\begin{aligned}
a_t &= \pi([x, a_{t-1}]) \\
&= \arg\min_a \|a - f([x_t, a_{t-1}])\|^2 + \lambda \|a - a_{t-1}\|^2 \\
&= \frac{f([x_t, a_{t-1}]) + \lambda a_{t-1}}{1+\lambda}
\end{aligned} \tag{C.7}$$

where $f \in \mathcal{F}$ is the minimizer of equation C.6

Thus we enforce smoothness of learning policy from Π_λ by encouraging low first order difference of consecutive actions of the executed trajectory $\{a_t\}$. In practice, we may contrain this first order difference relative to the human trajectory $\frac{1}{T}\sum_{t=1}^T \|a_t - a_{t-1}\| \leqslant \eta$, where $\eta \propto \frac{1}{T}\sum_{t=1}^T \left\|a_t^* - a_{t-1}^*\right\|$.

Consider any given iteration with the following set-up: we execute old policy $\pi = \pi_{old}$ to get rolled-out trajectory $\{a_t\}_{t=1}^T$. Form the new data set as $\mathcal{D} = \{(s_t, a_t^*)\}_{t=1}^T$

with predictors $s_t = [x_t, a_{t-1}]$ and feedback labels simply the human actions a_t^*. Use this data set to train a policy $\hat{\pi}$ by learning a supervised $\hat{f} \in \mathcal{F}$ from $\mathcal{D}$. Similar to π, the execution of $\hat{\pi}$ corresponds to $\hat{a}_t$ where:

$$\begin{aligned}\hat{a}_t &= \hat{\pi}([x_t, \hat{a}_{t-1}]) \\ &= \arg\min_a \left\| a - \hat{f}([x_t, \hat{a}_{t-1}]) \right\|^2 + \lambda \left\| a - \hat{a}_{t-1} \right\|^2 \\ &= \frac{\hat{f}([x_t, \hat{a}_{t-1}]) + \lambda \hat{a}_{t-1}}{1+\lambda}\end{aligned} \tag{C.8}$$

Denote by f_0 the "naive" supervised learner from $\mathcal{F}$. In other words, $f_0 = \arg\min_{f \in \mathcal{F}} \sum_{t=1}^{T} \|f([x_t, 0]) - a_t^*\|^2$. Let ω be the average gap between human trajectory and the rolled-out trajectory of f_0, i.e.

$$\omega = \frac{1}{T} \sum_{t=1}^{T} \|f_0([x_t, 0]) - a_t^*\|$$

Note that it is reasonable to assume that the average errors of f and $\hat{f}$ are no worse than f_0, since in the worst case we can simply discard the extra features a_{t-1} (resp. $\hat{a}_{t-1}$) of f (resp. $\hat{f}$) to recover the performance of the naive learner f_0:

$$\frac{1}{T} \sum_{t=1}^{T} \|f([x_t, a_{t-1}]) - a_t^*\| \leqslant \omega$$

$$\frac{1}{T} \sum_{t=1}^{T} \left\|\hat{f}([x_t, \hat{a}_{t-1}]) - a_t^*\right\| \leqslant \omega$$

Assume that the old policy $\pi = \pi_{old}$ is "bad" in the sense that the rolled-out trajectory $\{a_t\}_{t=1}^T$ differs substantially from human trajectory $\{a_t^*\}_{t=1}^T$. Specifically, denote the gap:

$$\frac{1}{T} \sum_{t=1}^{T} \|a_t^* - a_{t-1}\| = \Omega \gg \omega$$

This means the feedback correction a_t^* to $s_t = [x_t, a_{t-1}]$ is not smooth. We will show that the trained policy $\hat{\pi}$ from $\mathcal{D}$ will not be smooth.

From the definition of a_t and $\hat{a}_t$ from equations C.7 and C.8, we have for each t:

$$a_t - \hat{a}_t = \frac{\lambda}{1+\lambda}(a_{t-1} - \hat{a}_{t-1}) + \frac{f([x_t, a_{t-1}]) - \hat{f}([x_t, \hat{a}_{t-1}])}{1+\lambda}$$

Applying triangle inequality and summing up over t, we have:

$$\frac{1}{T} \sum_{t=1}^{T} \|a_t - \hat{a}_t\| \leqslant 2\omega$$

From here we can provide a lower bound on the smoothness of the new trajectory $\hat{a}_t$, as defined by the first order difference $\frac{1}{T}\sum_{t=1}^{T}\|\hat{a}_t - \hat{a}_{t-1}\|$. By definition of $\hat{a}_t$:

$$\begin{aligned}
\|\hat{a}_t - \hat{a}_{t-1}\| &= \left\| \frac{\hat{f}([x_t, \hat{a}_{t-1}]) - \hat{a}_{t-1}}{1+\lambda} \right\| \\
&= \left\| \frac{\hat{f}([x_t, \hat{a}_{t-1}]) - a_t^* + a_t^* - a_{t-1} + a_{t-1} - \hat{a}_{t-1}}{1+\lambda} \right\| \\
&\geqslant \frac{\|a_t^* - a_{t-1}\| - \left\|\hat{f}([x_t, \hat{a}_{t-1}]) - a_t^*\right\| - \|a_{t-1} - \hat{a}_{t-1}\|}{1+\lambda}
\end{aligned}$$

Again summing up over t and taking the average, we obtain:

$$\frac{1}{T}\sum_{t=1}^{T}\|\hat{a}_t - \hat{a}_{t-1}\| \geqslant \frac{\Omega - 3\omega}{1+\lambda}$$

Hence for $\Omega \gg \omega$, meaning the old trajectory is sufficiently far away from the ideal human trajectory, setting the learning target to be the ideal human actions will cause the learned trajectory to be non-smooth. □

C.2 Imitation Learning With Smooth Regression Forests

Variant of SIMILE Using Smooth Regression Forest Policy Class

We provide a specific instantiation of algorithm 4 that we used for our experiment, based on a policy class Π as a smooth regularized version of the space of tree-based ensembles. In particular, $\mathcal{F}$ is the space of random forests and $\mathcal{H}$ is the space of linear auto-regressors $\mathcal{H} \triangleq \{h(a_{t-1:t-\tau}) = \sum_{i=1}^{\tau} c_i a_{t-i}\}$. In combination, $\mathcal{F}$ and $\mathcal{H}$ form a complex tree-based predictor that can predict smooth sequential actions.

Empirically, decision tree-based ensembles are among the best performing supervised machine learning method (Caruana and Niculescu-Mizil, 2006; Criminisi et al., 2012). Due to the piece-wise constant nature of decision tree-based prediction, the results are inevitably non-smooth. We propose a recurrent extension based on $\mathcal{H}$, where the prediction at the leaf node is not necessarily a constant, but rather is a smooth function of both static leaf node prediction and its previous predictions. By merging the powerful tree-based policy class with a linear auto-regressor, we provide a novel approach to train complex models that can accommodate smooth sequential prediction using model-based smooth regularizer, at the same time leveraging the expressiveness of complex model-free function class (one can similarly apply the framework to the space of neural networks). Algorithm 19, which is based on SIMILE, describes in more details our training procedure used for the automated

Algorithm 19 Imitation Learning for Online Sequence Prediction with Smooth Regression Forest

Input: Input features $\mathbf{X} = \{x_t\}_{t=1}^T$, expert demonstration $\mathbf{A}^* = \{a_t^*\}_{t=1}^T$, base routine `Forest`, past horizon τ, sequence of $\sigma \in (0,1)$

Initialize $\mathbf{A}_0 \leftarrow \mathbf{A}^*, \mathbf{S}_0 \leftarrow \{\left[x_{t:t-\tau}, a^*_{t-1:t-\tau}\right]\}$,
$h_0 = \underset{c_1,\ldots,c_\tau}{\arg\min} \sum_{t=1}^{T} \left(a_t^* - \sum_{i=1}^{\tau} c_i a_{t-i}^*\right)^2$

Initial policy $\pi_0 = \hat{\pi}_0 \leftarrow$`Forest`$(\mathbf{S}_0, \mathbf{A}_0 |\ h_0)$

for $n = 1, \ldots, N$ **do**

  $\mathbf{A}_n = \{a_t^n\} \leftarrow \{\pi_{n-1}(\left[x_{t:t-\tau}, a^{n-1}_{t-1:t-\tau}\right])\}$ *//sequential roll-out old policy*

  $\mathbf{S}_n \leftarrow \{s_t^n = \left[x_{t:t-\tau}, a^n_{t-1:t-\tau}\right]\}$ *//Form states in 1d case*

  $\widehat{\mathbf{A}}_n = \{\widehat{a}_t^n = \sigma a_t^n + (1-\sigma) a_t^*\}\ \ \forall s_t^n \in \mathbf{S}_n$ *// collect smooth 1-step feedback*

  $h_n = \underset{c_1,\ldots,c_\tau}{\arg\min} \sum_{t=1}^{T} \left(\hat{a}_t^n - \sum_{i=1}^{\tau} c_i \hat{a}_{t-i}^n\right)^2$ *//update c_i via regularized least square*

  $\hat{\pi}_n \leftarrow$`Forest`$(\mathbf{S}_n, \widehat{\mathbf{A}}_n |\ h_n)$ *// train with smooth decision forests. See section C.2*

  $\beta \leftarrow \frac{\texttt{error}(\pi)}{\texttt{error}(\hat{\pi}) + \texttt{error}(\pi)}$ *//set β to weighted empirical errors*

  $\pi_n = \beta \hat{\pi}_n + (1-\beta)\pi_{n-1}$ *// update policy*

end for

output Last policy π_N

camera planning experiment. We first describe the role of the linear autoregressor class $\mathcal{H}$, before discussing how to incorporate $\mathcal{H}$ into decision tree training to make smooth prediction (see the next section).

The autoregresor $h_\pi(a_{-1}, \ldots, a_{-\tau})$ is typically selected from a class of autoregressors $\mathcal{H}$. In our experiments, we use regularized linear autoregressors as $\mathcal{H}$.

Consider a generic learning policy π with a rolled-out trajectory $\mathbf{A} = \{a_t\}_{t=1}^T$ corresponding to the input sequence $\mathbf{X} = \{x_t\}_{t=1}^T$. We form the state sequence $\mathbf{S} = \{s_t\}_{t=1}^T = \{[x_t, \ldots, x_{t-\tau}, a_{t-1}, \ldots, a_{t-\tau}]\}_{t=1}^T$, where τ indicates the past horizon that is adequate to approximately capture the full state information. We approximate the smoothness of the trajectory $\mathbf{A}$ by a linear autoregressor

$$h_\pi \equiv h_\pi(s_t) \equiv \sum_{i=1}^{\tau} c_i a_{t-i}$$

for a (learned) set of coefficients $\{c_i\}_{i=1}^\tau$ such that $a_t \approx h_\pi(s_t)$. Given feedback

target $\widehat{\mathbf{A}} = \{\hat{a}_t\}$, the joint loss function thus becomes

$$\begin{aligned}\ell(a, \hat{a}_t) &= \ell_d(a, \hat{a}_t) + \lambda \ell_R(a, s_t) \\ &= (a - \hat{a}_t)^2 + \lambda(a - \sum_{i=1}^{\tau} c_i a_{t-i})^2\end{aligned}$$

Here λ trades off between smoothness versus absolute imitation accuracy. The autoregressor h_π acts as a smooth linear regularizer, the parameters of which can be updated at each iteration based on feedback target $\widehat{\mathbf{A}}$ according to

$$\begin{aligned}h_\pi &= \arg\min_{h \in \mathcal{H}} \left\| \widehat{\mathbf{A}} - h(\widehat{\mathbf{A}}) \right\|^2 \\ &= \arg\min_{c_1,\ldots,c_\tau} (\sum_{t=1}^{T} (\hat{a}_t - \sum_{i=1}^{\tau} c_i \hat{a}_{t-i})^2), \end{aligned} \tag{C.9}$$

In practice we use a regularized version of equation (C.9) to learn a new set of coefficients $\{c_i\}_{i=1}^{\tau}$. The `Forest` procedure (Line 8 of algorithm 2) would use this updated h_π to train a new policy that optimizes the trade-off between $a_t \approx \hat{a}_t$ (feedback) versus smoothness as dictated by $a_t \approx \sum_{i=1}^{\tau} c_i a_{t-i}$.

Smooth Regularization with Linear Autoregressors

Our application of Algorithm 1 to realtime camera planning proceeds as follows: At each iteration, we form a state sequence $\mathbf{S}$ based on the rolled-out trajectory $\mathbf{A}$ and tracking input data $\mathbf{X}$ such that $s_t = [x_t, \ldots, x_{t-\tau}, a_{t-1}, \ldots, a_{t-\tau}]$ for appropriate τ that captures the history of the sequential decisions. We generate feedback targets $\widehat{\mathbf{A}}$ based on each $s_t \in \mathbf{S}$ following $\hat{a}_t = \sigma a_t + (1 - \sigma) a_t^*$ using a parameter $\sigma \in (0, 1)$ depending on the Euclidean distance between $\mathbf{A}$ and $\mathbf{A}^*$. Typically, σ gradually decreases to 0 as the rolled-out trajectory improves on the training set. After generating the targets, a new linear autoregressor h_π (new set of coefficients $\{c_i\}_{i=1}^{\tau}$) is learned based on $\widehat{\mathbf{A}}$ using regularized least squares (as described in the previous section). We then train a new model $\hat{\pi}$ based on $\mathbf{S}, \widehat{\mathbf{A}}$, and the updated coefficients $\{c_i\}$, using `Forest` - our recurrent decision tree framework that is capable of generating smooth predictions using autoregressor h_π as a smooth regularizer (see the following section for how to train smooth decision trees). Note that typically this creates a "chicken-and-egg" problem. As the newly learned policy $\hat{\pi}$ is greedily trained with respect to $\widehat{\mathbf{A}}$, the rolled-out trajectory of $\hat{\pi}$ may have a state distribution that is different from what the previously learned h_π would predict. Our approach offers two remedies to this circular problem. First, by allowing feedback signals

to vary smoothly relative to the current rolled-out trajectory $\mathbf{A}$, the new policy $\hat{\pi}$ should induce a new autoregresor that is similar to previously learned h_π. Second, by interpolating distributions (Line 10 of Algorithm 2) and having $\widehat{\mathbf{A}}$ eventually converge to the original human trajectory $\mathbf{A}^*$, we will have a stable and converging state distribution, leading to a stable and converging h_π.

Throughout iterations, the linear autoregressor h_π and regularization parameter λ enforces smoothness of the rolled-out trajectory, while the recurrent decision tree framework `Forest` learns increasingly accurate imitation policy. We generally achieve a satisfactory policy after 5-10 iterations in our sport broadcasting data sets. In the following section, we describe the mechanics of our recurrent decision tree training.

Smooth Regression Tree Training

Given states s as input, a decision tree specifies a partitioning of the input state space. Let $D = \{(s_m, \hat{a}_m)\}_{m=1}^{M}$ denote a training set of state/target pairs. Conventional regression tree learning aims to learn a partitioning such that each leaf node, `node`, makes a constant prediction via minimizing the squared loss function:

$$\begin{aligned} \bar{a}_{\texttt{node}} &= \arg\min_{a} \sum_{(s,\hat{a}) \in D_{\texttt{node}}} \ell_d(a, \hat{a}) \\ &= \arg\min_{a} \sum_{(s,\hat{a}) \in D_{\texttt{node}}} (\hat{a} - a)^2, \end{aligned} \tag{C.10}$$

where $D_{\texttt{node}}$ denotes the training data from D that has partitioned into the leaf `node`. For squared loss, we have:

$$\bar{a}_{\texttt{node}} = \texttt{mean}\left\{\hat{a} \,|(s, \hat{a}) \in D_{\texttt{node}}\right\}. \tag{C.11}$$

In the recurrent extension to `Forest`, we allow the decision tree to branch on the input state s, which includes the previous predictions $a_{-1}, \ldots, a_{-\tau}$. To enforce more explicit smoothness requirements, let $h_\pi(a_{-1}, \ldots, a_{-\tau})$ denote an autoregressor that captures the temporal dynamics of π over the distribution of input sequences $d_{\mathbf{x}}$, while *ignoring* the inputs x. At time step t, h_π predicts the behavior $a_t = \pi(s_t)$ given only $a_{t-1}, \ldots, a_{t-\tau}$.

Our policy class Π of recurrent decision trees π makes smoothed predictions by regularizing the predictions to be close to its autoregressor h_π. The new loss function incorporates both the squared distance loss ℓ_d, as well as a smooth regularization

loss such that:

$$\begin{aligned}\mathcal{L}_D(a) &= \sum_{(s,\hat{a})\in D} \ell_d(a,\hat{a}) + \lambda \ell_R(a,s) \\ &= \sum_{(s,\hat{a})\in D} (a-\hat{a})^2 + \lambda (y - h_\pi(s))^2\end{aligned}$$

where λ is a hyper-parameter that controls how much we care about smoothness versus absolute distance loss.

Making prediction: For any any tree/policy π, each leaf node is associated with the terminal leaf node value $\bar{a}_{\texttt{node}}$ such that prediction $\tilde{a}$ given input state s is:

$$\tilde{a}(s) \equiv \pi(s) = \arg\min_a \ (a - \bar{a}_{\texttt{node}(s)})^2 + \lambda (a - h_\pi(s))^2, \tag{C.12}$$

$$= \frac{\bar{a}_{\texttt{node}(s)} + \lambda h_\pi(s)}{1+\lambda}. \tag{C.13}$$

where $\texttt{node}(s)$ denotes the leaf node of the decision tree that s branches to.

Setting terminal node value: Given a fixed h_π and decision tree structure, navigating through consecutive binary queries eventually yields a terminal leaf node with associated training data $D_{\texttt{node}} \subset D$.

One option is to set the terminal node value $\bar{a}_{\texttt{node}}$ to satisfy:

$$\begin{aligned}\bar{a}_{\texttt{node}} &= \arg\min_a \sum_{(s,\hat{a})\in D_{\texttt{node}}} \ell_d(\tilde{a}(s|a),\hat{a}) \\ &= \arg\min_a \sum_{(s,\hat{a})\in D_{\texttt{node}}} (\tilde{a}(s|a) - \hat{a})^2 \\ &= \arg\min_a \sum_{(s,\hat{a})\in D_{\texttt{node}}} \left(\frac{a + \lambda h_\pi(s)}{1+\lambda} - \hat{a} \right)^2\end{aligned} \tag{C.14}$$

for $\tilde{a}(s|a)$ defined as in (C.13) with $a \equiv \bar{a}_{\texttt{node}(s)}$. Similar to (C.11), we can write the closed-form solution of (C.14) as:

$$\bar{a}_{\texttt{node}} = \texttt{mean}\left\{ (1+\lambda)\hat{a} - \lambda h_\pi(s) \,|(s,\hat{a}) \in D_{\texttt{node}} \right\}. \tag{C.15}$$

When $\lambda = 0$, (C.15) reduces to (C.11).

Note that (C.14) only looks at imitation loss ℓ_d, but not smoothness loss ℓ_R. Alternatively in the case of joint imitation and smoothness loss, the terminal leaf node is set to minimize the joint loss function:

$$
\begin{aligned}
\bar{a}_{\texttt{node}} &= \arg\min_{a} \mathcal{L}_{D_{\texttt{node}}}(\tilde{a}(s|a)) \\
&= \arg\min_{a} \sum_{(s,\hat{a})\in D_{\texttt{node}}} \ell_d(\tilde{a}(s|a),\hat{a}) + \lambda \ell_R(\tilde{a}(s|a), s) \\
&= \arg\min_{a} \sum_{(s,\hat{a})\in D_{\texttt{node}}} (\tilde{a}(s|a)-\hat{a})^2 + \lambda(\tilde{a}(s|a)-h_\pi(s))^2 \qquad \text{(C.16)} \\
&= \arg\min_{a} \sum_{(s,\hat{a})\in D_{\texttt{node}}} \left(\frac{a+\lambda h_\pi(s)}{1+\lambda} - \hat{a}\right)^2 \\
&+ \lambda \left(\frac{a+\lambda h_\pi(s)}{1+\lambda} - h_\pi(s)\right)^2 \\
&= \texttt{mean}\left\{\hat{a}\,|(s,\hat{a})\in D_{\texttt{node}}\right\}, \qquad \text{(C.17)}
\end{aligned}
$$

Node splitting mechanism: For a node representing a subset $D_{\texttt{node}}$ of the training data, the node impurity is defined as:

$$
\begin{aligned}
I_{\texttt{node}} &= \mathcal{L}_{D_{\texttt{node}}}(\bar{a}_{\texttt{node}}) \\
&= \sum_{(s,\hat{a})\in D_{\texttt{node}}} \ell_d(\bar{a}_{\texttt{node}},\hat{a}) + \lambda \ell_R(\bar{a}_{\texttt{node}}, s) \\
&= \sum_{(s,\hat{a})\in D_{\texttt{node}}} (\bar{a}_{\texttt{node}}-\hat{a})^2 + \lambda(\bar{a}_{\texttt{node}}-h_\pi(s))^2
\end{aligned}
$$

where $\bar{a}_{\texttt{node}}$ is set according to equation (C.15) or (C.17) over $(s,\hat{a})$'s in $D_{\texttt{node}}$. At each possible splitting point where $D_{\texttt{node}}$ is partitioned into $D_{\texttt{left}}$ and $D_{\texttt{right}}$, the impurity of the left and right child of the node is defined similarly. As with normal decision trees, the best splitting point is chosen as one that maximizes the impurity reduction: $I_{\texttt{node}} - \frac{|D_{\texttt{left}}|}{|D_{\texttt{node}}|} I_{\texttt{left}} - \frac{|D_{\texttt{right}}|}{|D_{\texttt{node}}|} I_{\texttt{right}}$

Appendix D

APPENDIX TO CHAPTER 5

D.1 Theoretical Analysis

Preliminaries and Notations

We formally define an ambient control policy space $\mathcal{U}$ to be a vector space equipped with inner product $\langle \cdot, \cdot \rangle : \mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$, which induces a norm $\|u\| = \sqrt{\langle u, u \rangle}$, and its dual norm defined as $\|v\|_* = \sup\{\langle v, u \rangle | \, \|u\| \leqslant 1\}$. While multiple ways to define the inner product exist, for concreteness we can think of the example of square-integrable stationary policies with $\langle u, v \rangle = \int_{\mathcal{S}} u(s)v(s)ds$. The addition operator $+$ between two policies $u, v \in \mathcal{U}$ is defined as $(u + v)(s) = u(s) + v(s)$ for all state $s \in \mathcal{S}$. Scaling $\lambda u + \kappa v$ is defined similarly for scalar λ, κ.

The cost functional of a control policy u is defined as $J(u) = \int_0^\infty c(s(\tau), u(\tau))d\tau$, or $J(u) = \int_{\mathcal{S}} c(s, u(s))d\mu^u(s)$, where μ^u is the distribution of states induced by policy u. This latter example is equivalent to the standard notion of value function in reinforcement learning.

Separate from the parametric representation issues, both programmatic policy class Π and neural policy class $\mathcal{F}$, and by extension - the joint policy class $\mathcal{H}$, are considered to live in the ambient vector space $\mathcal{U}$. We thus have a common and well-defined notion of distance between policies from different classes.

We make an important distinction between differentiability of $J(h)$ in the ambient policy space (non-parametric), versus differentiability in parameterization (parametric). For example, if Π is a class of decision-tree based policy, policies in Π may not be differentiable under representation. However, policies $\pi \in \Pi$ might still be differentiable when considered as points in the ambient vector space $\mathcal{U}$.

We will use the following standard notion of gradient and differentiability from functional analysis:

Definition D.1.1 (Subgradients)**.** The subgradient of J at h, denoted $\partial J(h)$, is the non-empty set $\{g \in \mathcal{H} | \forall j \in \mathcal{H} : \langle j - h, g \rangle + J(h) \leqslant J(j)\}$

Definition D.1.2 (Fréchet gradient)**.** A bounded linear operator $\nabla : \mathcal{H} \mapsto \mathcal{H}$ is called Fréchet functional gradient of J at $h \in \mathcal{H}$ if $\lim_{\|g\| \to 0} \frac{J(h+g) - J(h) - \langle \nabla J(h), g \rangle}{\|g\|} = 0$

The notions of convexity, smoothness and Bregman divergence are analogous to finite-dimensional setting:

Definition D.1.3 (Strong convexity)**.** A differentiable function R is $\alpha-$strongly convex w.r.t norm $\|\cdot\|$ if $R(y) \geqslant R(x) + \langle \nabla R(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2$

Definition D.1.4 (Lipschitz continuous gradient smoothness)**.** A differentiable function R is L_R-strongly smooth w.r.t norm $\|\cdot\|$ if $\|\nabla R(x) - \nabla R(y)\|_* \leqslant L_R \|x - y\|$

Definition D.1.5 (Bregman Divergence)**.** For a strongly convex regularizer R, $D_R(x, y) = R(x) - R(y) - \langle \nabla R(y), x - y \rangle$ is the Bregman divergence between x and y (not necessarily symmetric)

The following standard result for Bregman divergence will be useful:

Lemma D.1.1. *(Beck and Teboulle, 2003) For all x, y, z we have the identity $\langle \nabla R(x) - \nabla R(y), x - z \rangle = D_R(x, y) + D_R(z, x) - D_R(z, y)$. Since Bregman divergence is non-negative, a consequence of this identity is that $D_R(z, x) - D_R(z, y) \leqslant \langle \nabla R(x) - \nabla R(y), z - x \rangle$*

Expected Regret Bound under Noisy Policy Gradient Estimates and Projection Errors

In this section, we show regret bound for the performance of the sequence of returned programs $\pi_1, \ldots, \pi_T$ of the algorithm. The analysis here is agnostic to the particular implementation of algorithm 6 and algorithm 7.

Let R be a $\alpha-$strongly convex and L_R-smooth functional with respect to norm $\|\cdot\|$ on $\mathcal{H}$. The steps from algorithm 5 can be described as follows.

- Initialize $\pi_0 \in \Pi$. For each iteration t:
  1. Obtain a noisy estimate of the gradient $\widehat{\nabla} J(\pi_{t-1}) \approx \nabla J(\pi_{t-1})$
  2. Update in the $\mathcal{H}$ space: $\nabla R(h_t) = \nabla R(\pi_{t-1}) - \eta \widehat{\nabla} J(\pi_{t-1})$
  3. Obtain approximate projection $\pi_t = \textsc{Project}_{\pi}^{R}(h_t) \approx \arg\min_{pi \in \Pi} D_R(\pi, h_t)$

This procedure is an approximate functional mirror descent scheme under bandit feedback. We will develop the following result, which is a more detailed version of 5.4.1 in the main paper.

In the statement below, D is the diameter on Π with respect to defined norm $\|\cdot\|$ (i.e., $D = \sup \|\pi - \pi'\|$). L_J is the Lipschitz constant of the functional J on $\mathcal{H}$. β, σ^2 are the bound on the bias and variance of the gradient estimate at each iteration, respectively. α and L_R are the strongly convex and smooth coefficients of the functional regularizer R. Finally, ϵ is the bound on the projection error with respect to the same norm $\|\cdot\|$.

Theorem D.1.2 (Regret bound of returned policies)**.** *Let $\pi_1, \ldots, \pi_T$ be a sequence of programmatic policies returned by algorithm 5 and π^* be the optimal programmatic policy. We have the expected regret bound:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) \leqslant \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} + \beta D$$

In particular, choosing the learning rate $\eta = \sqrt{\frac{\frac{1}{T}+\epsilon}{\sigma^2}}$, the expected regret is simplified into:

(D.1)

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon} + \beta\right)$$

Proof. At each round t, let $\bar{\nabla}_t = \mathbb{E}[\widehat{\nabla}_t | \pi_t]$ be the conditional expectation of the gradient estimate. We will use the shorthand notation $\nabla_t = \nabla J(\pi_t)$. Denote the upper-bound on the bias of the estimate by β_t, i.e., $\left\|\bar{\nabla}_t - \nabla_t\right\|_* \leqslant \beta_t$ almost surely. Denote the noise of the gradient estimate by $\xi_t = \bar{\nabla}_t - \widehat{\nabla}_t$, and $\sigma_t^2 = \mathbb{E}\big[\left\|\widehat{\nabla}_t - \bar{\nabla}_t\right\|_*^2\big]$ is the variance of gradient estimate $\widehat{\nabla}_t$.

The projection operator is $\epsilon-$approximate in the sense that $\left\|\pi_t - \textsc{Project}_{\Pi}^{R}(f_t)\right\| = \left\|\widehat{\textsc{Project}}_{\Pi}^{R}(h_t) - \textsc{Project}_{\Pi}^{R}(h_t)\right\| \leqslant \epsilon$ with some constant ϵ, which reflects the statistical error of the imitation learning procedure. This projection error in general is independent of the choice of function classes Π and $\mathcal{F}$.We will use the shorthand notation $\pi_t^* = \textsc{Project}_{\Pi}^{R}(f_t)$ for the true Bregman projection of h_t onto Π.

Due to convexity of J over the space $\mathcal{H}$ (which includes Π), we have for all $\pi \in \Pi$:

$$J(\pi_t) - J(\pi) \leqslant \langle \nabla_t, \pi_t - \pi \rangle$$

We proceed to bound the RHS, starting with bounding the inner product where the actual gradient is replaced by the estimated gradient.

$$\langle \widehat{\nabla}_t, \pi_t - \pi \rangle = \frac{1}{\eta_t} \langle \nabla R(\pi_t) - \nabla R(h_{t+1}), \pi_t - \pi \rangle \tag{D.2}$$

$$= \frac{1}{\eta_t} \big(D_R(\pi, \pi_t) - D_R(\pi, h_{t+1}) + D_R(\pi_t, h_{t+1}) \big) \tag{D.3}$$

$$\leqslant \frac{1}{\eta_t} \big(D_R(\pi, \pi_t) - D_R(\pi, \pi^*_{t+1}) - D_R(\pi^*_{t+1}, h_{t+1}) + D_R(\pi_t, h_{t+1}) \big) \tag{D.4}$$

$$= \frac{1}{\eta_t} \big(\underbrace{D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1})}_{\text{telescoping}} + \underbrace{D_R(\pi, \pi_{t+1}) - D_R(\pi, \pi^*_{t+1})}_{\text{projection error}}$$

$$\underbrace{- D_R(\pi^*_{t+1}, h_{t+1}) + D_R(\pi_t, h_{t+1})}_{\text{relative improvement}} \big) \tag{D.5}$$

Equation (D.2) is due to the gradient update rule in $\mathcal{F}$ space. Equation (D.3) is derived from definition of Bregman divergence. Equation (D.4) is due to the generalized Pythagorean theorem of Bregman projection $D_R(x, y) \geqslant D_R(x, \textsc{Project}^R_\Pi(x)) + D_R(\textsc{Project}^R_\Pi(x), y)$. The RHS of equation (D.4) are decomposed into three components that will be bounded separately.

Bounding projection error. By lemma (D.1.1) we have

$$D_R(\pi, \pi_{t+1}) - D_R(\pi, \pi^*_{t+1}) \leqslant \langle \nabla R(\pi_{t+1}) - \nabla R(\pi^*_{t+1}), \pi - \pi_{t+1} \rangle \tag{D.6}$$

$$\leqslant \left\| \nabla R(\pi_{t+1}) - \nabla R(\pi^*_{t+1}) \right\| \left\| \pi - \pi_{t+1} \right\|_* \tag{D.7}$$

$$\leqslant L_R \left\| \pi_{t+1} - \pi^*_{t+1} \right\| D \leqslant \epsilon L_R D \tag{D.8}$$

Equation (D.7) is due to Cauchy–Schwarz. Equation (D.8) is due to Lipschitz smoothness of ∇R and definition of $\epsilon-$approximate projection.

Bounding relative improvement. This follows standard argument from analysis of mirror descent algorithm.

$$D_R(\pi_t, h_{t+1}) - D_R(\pi^*_{t+1}, h_{t+1}) = R(\pi_t) - R(\pi^*_{t+1}) + \langle \nabla R(h_{t+1}), \pi^*_{t+1} - \pi_t \rangle \tag{D.9}$$

$$\leqslant \langle \nabla R(\pi_t), \pi_t - \pi^*_{t+1} \rangle - \frac{\alpha}{2} \left\| \pi^*_{t+1} - \pi_t \right\|^2_* + \langle \nabla R(h_{t+1}), \pi^*_{t+1} - \pi_t \rangle \tag{D.10}$$

$$= -\eta_t \langle \widehat{\nabla}_t, \pi^*_{t+1} - \pi_t \rangle - \frac{\alpha}{2} \left\| \pi^*_{t+1} - \pi_t \right\|^2 \tag{D.11}$$

$$\leqslant \frac{\eta_t^2}{2\alpha} \left\| \widehat{\nabla}_t \right\|^2_* \leqslant \frac{\eta_t^2}{\alpha} (\sigma_t^2 + L_J^2) \tag{D.12}$$

Equation (D.10) is from the $\alpha-$strong convexity property of regularizer R. Equation (D.11) is by definition of the gradient update. Combining the bounds on the three components and taking expectation, we thus have

$$\mathbb{E}\left[\langle \widehat{\nabla}_t, \pi_t - \pi \rangle\right] \leqslant \frac{1}{\eta_t}\left(D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1}) + \epsilon L_R D + \frac{\eta_t^2}{\alpha}(\sigma_t^2 + L_J^2)\right) \tag{D.13}$$

Next, the difference between estimated gradient $\widehat{\nabla}_t$ and actual gradient ∇_t factors into the bound via Cauchy-Schwarz:

$$\mathbb{E}\left[\langle \nabla_t - \widehat{\nabla}_t, \pi_t - \pi \rangle\right] \leqslant \left\|\nabla_t - \mathbb{E}[\widehat{\nabla}_t]\right\|_* \|\pi_t - \pi\| \leqslant \beta_t D \tag{D.14}$$

The results can be deduced from equations (D.13) and (D.14).

Unbiased gradient estimates. For the case when the gradient estimate is unbiased, assume the variance of the noise of gradient estimates is bounded by σ^2, we have the expected regret bound for all $pi \in \Pi$

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi) \leqslant \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} \tag{D.15}$$

here to clarify, L_R is the smoothness coefficient of regularizer R (i.e., the gradient of R is L_R-Lipschitz, L_J is Lipschitz constant of J, D is the diameter of Π under norm $\|\cdot\|$, σ^2 is the upper-bound on the variance of gradient estimates, and ϵ is the error from the projection procedure (i.e., imitation learning loss).

We can set learning rate $\eta = \sqrt{\frac{\frac{1}{T}+\epsilon}{\sigma^2}}$ to observe that the expected regret is bounded by $O(\sigma\sqrt{\frac{1}{T}+\epsilon})$.

Biased gradient estimates. Assume that the bias of gradient estimate at each round is upper-bounded by $\beta_t \leqslant \beta$. Similar to before, combining inequalities from (D.13) and (D.14), we have

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi) \leqslant \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} + \beta D \tag{D.16}$$

Similar to before, we can set learning rate $\eta = \sqrt{\frac{\frac{1}{T}+\epsilon}{\sigma^2}}$ to observe that on the expected regret is bounded by $O(\sigma\sqrt{\frac{1}{T}+\epsilon} + \beta)$. Compared to the bound on (D.15), in the biased case, the extra regret incurred per bound is simply a constant, and does not depend on T. □

Finite-Sample Analysis

In this section, we provide overall finite-sample analysis for PROPEL under some simplifying assumptions. We first consider the case where exact gradient estimate is available, before extending the result to the general case of noisy policy gradient update. Combining the two steps will give us the proof for the following statement (theorem 5.4.2 in the main paper)

Theorem D.1.3 (Finite-sample guarantee)**.** *At each iteration, we perform vanilla policy gradient estimate of π (over $\mathcal{H}$) using m trajectories and use DAgger algorithm to collect M roll-outs. Setting the learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}\right)}$, after T rounds of the algorithm, we have that*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leqslant O\left(\sigma\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right) + O\left(\sigma\sqrt{\frac{\log(Tk/\delta)}{m}} + \frac{AH\log(Tk/\delta)}{m}\right)$$

holds with probability at least $1-\delta$, with H the task horizon, A the cardinality of action space, σ^2 the variance of policy gradient estimates, and k the dimension Π's parameterization.

Exact gradient estimate case. Assuming that the policy gradients can be calculated exactly, it is straight-forward to provide high-probability guarantee for the effect of the projection error. We start with the following result, adapted from (Ross et al., 2011b) for the case of projection error bound. In this version of DAgger, we assume that we only collect a single *(state, expert action)* pair from each trajectory roll-out. Result is similar, with tighter bound, when multiple data points are collected along the trajectory.

Lemma D.1.4 (Projection error bound from imitation learning procedure)**.** *Using DAgger as the imitation learning sub-routine for our PROJECT operator in algorithm 7, let M be the number of trajectories rolled-out for learning, and H be the horizon of the task. With probability at least $1-\delta$, we have*

$$D_R(\pi, \pi^*) \leqslant \widetilde{O}(1/M) + \frac{2\ell_{max}(1+H)}{M} + \sqrt{\frac{2\ell_{max}\log(1/\delta))}{M}}$$

where π is the result of PROJECT, π^* is the true Bregman projection of h onto Π, and ℓ_{max} is the maximum value of the imitation learning loss function $D_R(\cdot,\cdot)$

The bound in lemma D.1.4 is simpler than previous imitation learning results with cost information ((Ross and Bagnell, 2014; Ross et al., 2011b). The reason is that the goal of the PROJECT operator is more modest. Since we only care about the distance between the empirical projection π and the true projection π^*, the loss objective in imitation learning is simplified (i.e., this is only a regret bound), and we can disregard how well policies in Π can imitate the expert h, as well as the performance of $J(\pi)$ relative to the true cost from the environment $J(h)$.

A consequence of this lemma is that for the number of trajectories at each round of imitation learning $M = O(\frac{\log 1/\delta}{\epsilon^2}) + O(\frac{H}{\epsilon})$, we have $D_R(\pi_t, \pi_t^*) \leqslant \epsilon$ with probability at least $1-\delta$. Applying union bound across T rounds of learning, we obtain the following guarantee (under no gradient estimation error)

Proposition D.1.5 (Finite-sample Projection Error Bound)**.** *To simplify the presentation of the result, we consider L_R, D, L, α to be known constants. Using DAgger algorithm to collect $M = O(\frac{\log T/\delta}{\epsilon^2}) + O(\frac{H}{\epsilon})$ roll-outs at each iteration, we have the following regret guarantee after T rounds of our main algorithm:*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leqslant O\left(\frac{1}{\eta T} + \frac{\epsilon}{\eta} + \eta\right)$$

with probability at least $1-\delta$. Consequently, setting $\eta = \sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}$, we have that

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leqslant O\left(\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right)$$

with probability at least $1-\delta$

Note that the dependence on the time horizon of the task is sub-linear. This is different from standard imitation learning regret bounds, which are often at least linear in the task horizon. The main reason is that our comparison benchmark π^* does live in the space Π, whereas for DAgger, the expert policy may not reside in the same space.

Noisy gradient estimate case. We now turn to the issue of estimating the gradient of $\nabla J(\pi)$. We make the following simplifying assumption about the gradient estimation:

- The π is parameterized by vector $\theta \in \mathbb{R}^k$ (such as a neural network). The parameterization is differentiable with respect to θ (Alternatively, we can view Π as a differentiable subspace of $\mathcal{F}$, in which case we have $\mathcal{H} = \mathcal{F}$)
- At each UPDATE loop, the policy is rolled out m times to collect the data, each trajectory has horizon length H
- For each visited state $s \sim d_h$, the policy takes a uniformly random action a. The action space is finite with cardinality A.
- The gradient ∇h_θ is bounded by B

The gradient estimate is performed consistent with a generic policy gradient scheme, i.e.,

$$\widehat{\nabla} J(\theta) = \frac{A}{m} \sum_{i=1}^{H} \sum_{j=1}^{m} \nabla \pi_\theta(a_i^j | s_i^j, \theta) \widehat{Q}_i^j$$

where $\widehat{Q}_i^j$ is the estimated cost-to-go (Sutton et al., 2000).

Taking uniform random exploratory actions ensures that the samples are i.i.d. We can thus apply Bernstein's inequality to obtain the bound between estimated gradient and the true gradient. Indeed, with probability at least $1 - \delta$, we have that the following bound on the bias component-wise:

$$\left\| \widehat{\nabla} J(\theta) - \nabla J(\theta) \right\|_\infty \leqslant \beta \text{ when } m \geqslant \frac{(2\sigma^2 + 2AHB\frac{\beta}{3}) \log \frac{k}{\delta}}{\beta^2}$$

which leads to similar bound with respect to $\|\cdot\|_*$ (here we leverage the equivalence of norms in finite dimensional setting):

$$\left\| \nabla_t - \widehat{\nabla}_t \right\|_* \leqslant \beta \text{ when } m = O\left(\frac{(\sigma^2 + AHB\beta) \log \frac{k}{\delta}}{\beta^2} \right)$$

Applying union bound of this result over T rounds of learning, and combining with the result from proposition (D.1.5), we have the following finite-sample guarantee in the simplifying policy gradient update. This is also the more detailed statement of theorem 5.4.2 in the main paper.

Proposition D.1.6 (Finite-sample Guarantee under Noisy Gradient Updates and Projection Error)**.** *At each iteration, we perform policy gradient estimate using* $m = O(\frac{(\sigma^2 + AHB\beta) \log \frac{Tk}{\delta}}{\beta^2})$ *trajectories and use DAgger algorithm to collect* $M =$

$O(\frac{\log T/\delta}{\epsilon^2}) + O(\frac{H}{\epsilon})$ *roll-outs. Setting the learning rate* $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}\right)}$, *after* T *rounds of the algorithm, we have that*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leqslant O\left(\sigma\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right) + \beta$$

with probability at least $1-\delta$.

Consequently, we also have the following regret bound:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leqslant & O\left(\sigma\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right) \\ & + O\left(\sigma\sqrt{\frac{\log(Tk/\delta)}{m}} + \frac{AH\log(Tk/\delta)}{m}\right)\end{aligned}$$

holds with probability at least $1-\delta$, *where again* H *is the task horizon,* A *is the cardinality of action space, and* k *is the dimension of function class* Π*'s parameterization.*

Proof. (For both proposition (D.1.6) and (D.1.5)). The results follow by taking the inequality from equation (D.16), and by solving for ϵ and β explicitly in terms of relevant quantities. Based on the specification of M and m, we obtain the necessary precision for each round of learning in terms of number of trajectories:

$$\begin{aligned}\beta &= O(\sigma\frac{\log(k/\delta)}{m} + \frac{AHB\log(k/\delta)}{m}) \\ \epsilon &= O(\frac{H}{M} + \sqrt{\frac{\log(1/\delta)}{M}})\end{aligned}$$

Setting the learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \epsilon\right)}$ and rearranging the inequalities lead to the desired bounds. □

The regret bound depends on the variance σ^2 of the policy gradient estimates. It is well-known that vanilla policy gradient updates suffer from high variance. We instead use functional regularization technique, based on CORE-RL, in the practical implementation of our algorithm. The CORE-RL subroutine has been demonstrated to reduce the variance in policy gradient updates (Cheng et al., 2019e).

Defining a consistent approximation of $\nabla_{\mathcal{H}} J(\pi)$ - Proof of Proposition 5.4.3
We are using the notion of Fréchet derivative to define gradient of differentiable functional. Note that while Gateaux derivative can also be utilized, Fréchet derivative ensures continuity of the gradient operator that would be useful for our analysis.

Definition D.1.6 (Fréchet gradient)**.** A bounded linear operator $\nabla : \mathcal{H} \mapsto \mathcal{H}$ is called Fréchet functional gradient of J at $h \in \mathcal{H}$ if $\lim_{\|g\| \to 0} \frac{J(h+g) - J(h) - \langle \nabla J(h), g \rangle}{\|g\|} = 0$

We make the following assumption about $\mathcal{H}$ and $\mathcal{F}$. One interpretation of this assumption is that the space of policies Π and $\mathcal{F}$ that we consider have the property that a programmatic policy $\pi \in \Pi$ can be well-approximated by a large space of neural policies $f \in \mathcal{F}$.

Assumption 6. *J is Fréchet differentiable on $\mathcal{H}$. J is also differentiable on the restricted subspace $\mathcal{F}$. And $\mathcal{F}$ is dense in $\mathcal{H}$ (i.e., the closure $\bar{\mathcal{F}} = \mathcal{H}$)*

It is then clear that $\forall\ f \in \mathcal{F}$ the Fréchet gradient $\nabla_{\mathcal{F}} J(f)$, restricted to the subspace $\mathcal{F}$ is equal to the gradient of f in the ambient space $\mathcal{H}$ (since Fréchet gradient is unique). In general, given $\pi \in \Pi$ and $f \in \mathcal{F}$, $\pi + f$ is not necessarily in $\mathcal{F}$. However, the restricted gradient on subspace $\mathcal{F}$ of $J(\pi + f)$ can be defined asymptotically.

Proposition D.1.7. *Fixing a policy $\pi \in \Pi$, define a sequence of policies $f_k \in \mathcal{F}$, $k = 1, 2, \ldots$ that converges to π: $\lim_{k \to \infty} \|f_k - g\| = 0$, we then have $\lim_{k \to \infty} \|\nabla_{\mathcal{F}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$*

Proof. Since Fréchet derivative is a continuous linear operator, we have $\lim_{k \to \infty} \|\nabla_{\mathcal{H}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$. By the reasoning above, for $f \in \mathcal{F}$, the gradient $\nabla_{\mathcal{F}} J(f)$ defined via restriction to the space $\mathcal{F}$ does not change compared to $\nabla_{\mathcal{H}} J(f)$, the gradient defined over the ambient space $\mathcal{H}$. Thus we also have $\lim_{k \to \infty} \|\nabla_{\mathcal{F}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$. By the same argument, we also have that for any given $\pi \in \Pi$ and $f \in \mathcal{F}$, even if $\pi + f \notin \mathcal{F}$, the gradient $\nabla_{\mathcal{F}} J(\pi + f)$ with respect to the $\mathcal{F}$ can be approximated similarly. □

Note that we are not assuming $J(\pi)$ to be differentiable when restricting to the policy subspace Π.

Theoretical motivation for Algorithm 6 - Proof of Proposition 5.4.4 and 5.4.5
We consider the case where Π is not differentiable by parameterization. Note that this does not preclude $J(\pi)$ for $\pi \in \Pi$ to be differentiable in the non-parametric function space. Two complications arise compared to our previous approximate mirror descent procedure. First, for each $\pi \in \Pi$, estimating the gradient $\nabla J(\pi)$ (which may not exist under certain parameterization, per section 5.4) can become much more difficult. Second, the update rule $\nabla R(\pi) - \nabla_{\mathcal{F}} J(\pi)$ may not be in the dual space of $\mathcal{F}$, as in the simple case where $\Pi \subset \mathcal{F}$, thus making direct gradient update in the $\mathcal{F}$ space inappropriate.

Assumption 7. *J is convex in $\mathcal{H}$.*

By convexity of J in $\mathcal{H}$, sub-gradients $\partial J(h)$ exists for all $h \in \mathcal{H}$. In particular, $\partial J(\pi)$ exists for all $\pi \in \Pi$. Note that $\partial J(\pi)$ reflects sub-gradient of π with respect to the ambient policy space $\mathcal{H}$.

We will make use of the following equivalent perspective to mirror descent(Beck and Teboulle, 2003), which consists of two-step process for each iteration t

1. Solve for $h_{t+1} = \arg\min_{h \in \mathcal{H}} \eta \langle \partial J(\pi_t), h \rangle + D_R(h, \pi_t)$
2. Solve for $\pi_{t+1} = \arg\min_{\pi \in \Pi} D_R(\pi, h_{t+1})$

We will show how this version of the algorithm motivates our main algorithm. Consider step 1 of the main loop of PROPEL, where given a fixed $\pi \in \Pi$, the optimization problem within $\mathcal{H}$ is

$$(\text{OBJECTIVE\_1}) = \min_{h \in \mathcal{H}} \eta \langle \partial J(\pi), h \rangle + D_R(h, \pi) \tag{D.17}$$

Due to convexity of $\mathcal{H}$ and the objective, problem (OBJECTIVE_1) is equivalent to:

$$(\text{OBJECTIVE\_1}) = \min \langle \partial J(\pi), h \rangle \tag{D.18}$$

$$\text{s.t. } D_R(h, \pi) \leqslant \tau \tag{D.19}$$

where τ depends on η. Since π is fixed, this optimization problem can be relaxed by choosing $\lambda \in [0, 1]$, and a set of candidate policies $h = \pi + \lambda f$, for all $f \in \mathcal{F}$, such that $D_R(h, \pi) \leqslant \tau$ is satisfied (Selection of λ is possible with bounded spaces). Since this constraint set is potentially a restricted set compared to the space of

policies satisfying inequality (D.19), the optimization problem (D.17) is relaxed into:

$$(\text{OBJECTIVE\_1}) \leqslant (\text{OBJECTIVE\_2}) = \min_{f \in \mathcal{F}} \langle \partial J(\pi), \pi + \lambda f \rangle \tag{D.20}$$

Due to convexity property of J, we have

$$\langle \partial J(\pi), \lambda f \rangle = \langle \partial J(\pi), \pi + \lambda f - \pi) \rangle \leqslant J(\pi + \lambda f) - J(\pi) \tag{D.21}$$

The original problem OBJECTIVE_1 is thus upper bounded by:

$$\min_{h \in \mathcal{H}} \eta \langle \partial J(\pi), h) \rangle + D_R(h, \pi) \leqslant \min_{f \in \mathcal{F}} J\big(\pi + \lambda f\big) - J(\pi) + \langle \partial J(\pi), \pi \rangle$$

Thus, a relaxed version of original optimization problem OBJECTIVE_1 can be obtained by miniziming $J(\pi + \lambda f)$ over $f \in \mathcal{F}$ (note that π is fixed). This naturally motivates using functional regularization technique, such as CORE-RL algorithm (Cheng et al., 2019e), to update the parameters of differentiable function f via policy gradient descent update:

$$f' = f - \eta \lambda \nabla_{\mathcal{F}} \lambda J(\pi + \lambda f)$$

where the gradient of J is taken with respect to the parameters of f (neural networks). This is exactly the update step in algorithm 6 (also similar to iterative updte of CORE-RL algorithm), where the neural network policy is regularized by a prior controller π.

Statement and Proof of Proposition 5.4.5

Proposition D.1.8 (Regret bound for the relaxed optimization objective)**.** *Assuming $J(h)$ is L-strongly smooth over $\mathcal{H}$, i.e., $\nabla_{\mathcal{H}} J(h)$ is L-Lipschitz continuous, approximating UPDATE$_{\mathcal{H}}$ by UPDATE$_F$ per Alg. 6 leads to the expected regret bound:* $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\lambda\sigma\sqrt{\frac{1}{T} + \epsilon} + \lambda^2 L^2\right)$

Proof. Instead of focusing on the bias of the gradient estimate $\nabla_{\mathcal{H}} J(\pi)$, we will shift our focus on the alternative proximal formulation of mirror descent, under optimization and projection errors. In particular, at each iteration t, let $h^*_{t+1} = \arg\min_{h \in \mathcal{H}} \eta \langle \nabla J(\pi_t), h \rangle + D_R(h, \pi_t)$ and let the optimization error be defined as β_t where $\nabla R(h_{t+1}) = \nabla R(h^*_{t+1}) + \beta_t$. Note here that this is different from (but related to) the notion of bias from gradient estimate of $\nabla J(\pi)$ used in theorem 5.4.1 and theorem D.1.2. The projection error from imitation learning procedure is defined

similarly to theorem 5.4.1: $\pi^*_{t+1} = \arg\min_{\pi\in\Pi} D_R(\pi, h_{t+1})$ is the true projection, and $\left\|\pi_{t+1} - \pi^*_{t+1}\right\| \leqslant \epsilon$.

We start with similar bounding steps to the proof of theorem 5.4.1:

$$\begin{aligned}\langle \nabla J(\pi_t), \pi_t - \pi\rangle &= \frac{1}{\eta}\langle \nabla R(h^*_{t+1}) - \nabla R(\pi_t), \pi_t - \pi\rangle \\ &= \frac{1}{\eta}\left(\langle \nabla R(h_{t+1}) - \nabla R(\pi_t), \pi_t - \pi\rangle - \langle \beta_t, \pi_t - \pi\rangle\right) \\ &= \underbrace{\frac{1}{\eta}\left(D_R(\pi, \pi_t) - D_R(\pi, h_{t+1}) + D_R(\pi_t, h_{t+1})\right)}_{\text{component\_1}} + \underbrace{\frac{1}{\eta}\langle \beta_t, \pi_t - \pi\rangle}_{\text{component\_2}}\end{aligned} \tag{D.22}$$

As seen from the proof of theorem D.1.2, component_1 can be upperbounded by:

$$\frac{1}{\eta}\Big(\underbrace{D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1})}_{\text{telescoping}} + \underbrace{D_R(\pi, \pi_{t+1}) - D_R(\pi, \pi^*_{t+1})}_{\text{projection error}} \underbrace{- D_R(\pi^*_{t+1}, h_{t+1}) + D_R(\pi_t, h_{t+1})}_{\text{relative improvement}}\Big)$$

The bound on projection error is identical to theorem D.1.2:

$$D_R(\pi, \pi_t) - D_R(\pi, \pi^*_{t+1}) \leqslant \epsilon L_R D \tag{D.23}$$

The bound on relative improvement is slightly different:

$$\begin{aligned}&D_R(\pi_t, h_{t+1}) - D_R(\pi^*_{t+1}, h_{t+1}) = R(\pi_t) - R(\pi^*_{t+1}) + \langle \nabla R(h_{t+1}), \pi^*_{t+1} - \pi_t\rangle \\ &= R(\pi_t) - R(\pi^*_{t+1} + \langle \nabla R(h^*_{t+1}), \pi^*_{t+1} - \pi_t\rangle) + \langle \beta_t, \pi^*_{t+1} - \pi_t\rangle \\ &\leqslant \langle \nabla R(\pi_t), \pi_t - \pi^*_{t+1}\rangle - \frac{\alpha}{2}\left\|\pi^*_{t+1} - \pi_t\right\|^2 \\ &\qquad + \langle \nabla R(h^*_{t+1}), \pi^*_{t+1} - \pi_t\rangle + \langle \beta_t, \pi^*_{t+1} - \pi_t\rangle \\ &= -\eta\langle \nabla J_{\mathcal{H}}(\pi_t), \pi^*_{t+1} - \pi_t\rangle - \frac{\alpha}{2}\left\|\pi^*_{t+1} - \pi_t\right\|^2 + \langle \beta_t, \pi^*_{t+1} - \pi_t\rangle \qquad \text{(D.24)} \\ &\leqslant \frac{\eta^2}{2\alpha}\left\|\nabla_{\mathcal{H}} J(\pi_t)\right\|^2_* + \langle \beta_t, \pi^*_{t+1} - \pi_t\rangle \\ &\leqslant \frac{\eta^2}{2\alpha}L_J^2 + \langle \beta_t, \pi^*_{t+1} - \pi_t\rangle \qquad \text{(D.25)}\end{aligned}$$

Note here that the gradient $\nabla_{\mathcal{H}} J(\pi_t)$ is not the result of estimation. Combining equations (D.22), (D.23), (D.24), (D.25), we have:

$$\langle \nabla J(\pi_t), \pi_t - \pi\rangle \leqslant \frac{1}{\eta}\Big(D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1}) + \epsilon L_R D + \frac{\eta^2}{2\alpha}L_J^2 + \langle \beta_t, \pi^*_{t+1} - \pi\rangle\Big) \tag{D.26}$$

Next, we want to bound β_t. Choose regularizer R to be $\frac{1}{2}\|\cdot\|^2$ (consistent with the pseudocode in algorithm 6). We have that:

$$h^*_{t+1} = \arg\min_{h\in\mathcal{H}} \eta\langle\nabla J(\pi_t), h\rangle + \frac{1}{2}\|h-\pi_t\|^2$$

which is equivalent to:

$$h^*_{t+1} = \pi_t + \arg\min_{f\in\mathcal{F}} \eta\langle\nabla J(\pi_t), f\rangle + \frac{1}{2}\|f\|^2$$

Let $f^*_{t+1} = \arg\min_{f\in\mathcal{F}} \eta\langle\nabla J(\pi_t), f\rangle + \frac{1}{2}\|f\|^2$. Taking the gradient over f, we can see that $f^*_{t+1} = -\eta\nabla J(\pi_t)$. Let f_{t+1} be the minimizer of $\min_{f\in\mathcal{F}} J(\pi_t+\lambda f)$. We then have $h^*_{t+1} = \pi_t + f^*_{t+1}$ and $h_{t+1} = \pi + \lambda f_{t+1}$. Thus $\beta_t = h_{t+1} - h^*_{t+1} = f_{t+1} - f^*_{t+1}$.

On one hand, we have

$$J(\pi_t+\lambda f_{t+1}) \leqslant J(\pi_t+\omega f^*_{t+1}) \leqslant J(\pi_t) + \langle\nabla J(\pi_t), \omega f^*_{t+1}\rangle + \frac{L}{2}\left\|\omega f^*_{t+1}\right\|^2$$

due to optimality of f_{t+1} and strong smoothness property of J. On the other hand, since J is convex, we also have the first-order condition:

$$J(\pi_t+\lambda f_{t+1}) \geqslant J(\pi_t) + \langle\nabla J(\pi_t), \lambda f_{t+1}\rangle$$

Combine with the inequality above, and subtract $J(\pi_t)$ from both sides, and using the relationship $f^*_{t+1} = -\eta\nabla J(\pi_t)$, we have that:

$$\langle -\frac{1}{\eta} f^*_{t+1}, \lambda f_{t+1}\rangle \leqslant \langle -\frac{1}{\eta} f^*_{t+1}, \omega f^*_{t+1}\rangle + \frac{L\omega^2}{2}\left\|f^*_{t+1}\right\|^2$$

Since this is true $\forall\omega$, rearrange and choose ω such that $\frac{\omega}{\eta} - \frac{L\omega^2}{2} = -\frac{\lambda}{2\eta}$, namely $\omega = \frac{1-\sqrt{1-\lambda\eta L}}{L\eta}$, and complete the square, we can establish the bound that:

$$\left\|f_{t+1} - f^*_{t+1}\right\| \leqslant \eta(\lambda L)^2 B \tag{D.27}$$

for B the upperbound on $\|f_{t+1}\|$. We thus have $\|\beta_t\| = O(\eta(\lambda L)^2)$. Plugging the result from equation D.27 into RHS of equation D.26, we have:

$$\langle\nabla J(\pi_t), \pi_t - \pi\rangle \leqslant \frac{1}{\eta}\big(D_R(\pi,\pi_t) - D_R(\pi,\pi_{t+1}) + \epsilon L_R D + \frac{\eta^2}{2\alpha}L_J^2\big) + \big(\eta(\lambda L)^2 B\big) \tag{D.28}$$

Since J is convex in $\mathcal{H}$, we have $J(\pi_t) - J(\pi) \leqslant \langle\nabla J(\pi_t), \pi_t - \pi\rangle$. Similar to theorem 5.4.1, setting $\eta = \sqrt{\frac{1}{\lambda^2\sigma^2}(\frac{1}{T}+\epsilon)}$ and taking expectation on both sides, we have:

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\big(\lambda\sigma\sqrt{\frac{1}{T}+\epsilon} + \lambda^2 L^2\big) \tag{D.29}$$

Note that unlike regret bound from theorem 5.4.1 under general bias, variance of gradient estimate and projection error, σ^2 here explicitly refers to the bound on neural-network based policy gradient variance. The variance reduction of $\lambda\sigma$, at the expense of some bias, was also similarly noted in a recent functional regularization technique for policy gradient (Cheng et al., 2019e). □

D.2 Additional Experimental Results and Details

TORCS

We generate controllers for cars in *The Open Racing Car Simulator* (TORCS) (Wymann et al., 2014). In its full generality TORCS provides a rich environment with input from up to 89 sensors, and optionally the 3D graphic from a chosen camera angle in the race. The controllers have to decide the values of 5 parameters during game play, which correspond to the acceleration, brake, clutch, gear and steering of the car.

Apart from the immediate challenge of driving the car on the track, controllers also have to make race-level strategy decisions, like making pit-stops for fuel. A lower level of complexity is provided in the Practice Mode setting of TORCS. In this mode all race-level strategies are removed. Currently, so far as we know, state-of-the-art DRL models are capable of racing only in Practice Mode, and this is also the environment that we use. Here we consider the input from 29 sensors, and decide values for the acceleration, steering, and braking actions.

We chose a suite of tracks that provide varying levels of difficulty for the learning algorithms. In particular, for the tracks Ruudskogen and Alpine-2, the DDPG agent is unable to reliably learn a policy that would complete a lap. We perform the experiments with twenty-five random seeds and report the median lap time over these twenty-five trials. However we note that the TORCS simulator is not deterministic even for a fixed random seed. Since we model the environment as a Markov Decision Process, this non-determinism is consistent with our problem statement.

For our Deep Reinforcement Learning agents we used standard open source implementations (with pre-tuned hyper-parameters) for the relevant domain.

All experiments were conducted on standard workstation with a 2.5 GHz Intel Core i7 CPU and a GTX 1080 Ti GPU card.

The code for the TORCS experiments can be found at:
https://bitbucket.org/averma8053/propel

In Table D.1 we show the lap time performance and crash ratios of PROPEL agents initialized with neural policies obtained via DDPG. As discussed in Section 5.5, DDPG often exhibits high variance across trials and this adversely affects the performance of the PROPEL agents when they are initialized via DDPG. In Table D.2 we show generalization results for the IPPGTREE agent. As noted in Section 5.5, the generalization results for IPPGTREE are in between those of DDPG and IPPGPROGRAM.

Verified Smoothness Property. For the program given in Figure 2 we proved using symbolic verification techniques, that $\forall k$,
$\sum_{i=k}^{k+5} \|\textbf{peek}(s[\texttt{RPM}], i+1) - \textbf{peek}(s[\texttt{RPM}], i)\| < 0.003$
$\implies \|\textbf{peek}(a[\texttt{Accel}], k+1) - \textbf{peek}(a[\texttt{Accel}], k)\| < 0.63$. Here the function $\textbf{peek}(., i)$ takes in a history/sequence of sensor or action values and returns the value at position i, . Intuitively, the above logical implication means that if the sum of the consecutive differences of the last six `RPM` sensor values is less than 0.003, then the acceleration actions calculated at the last and penultimate step will not differ by more than 0.63.

Table D.1: *Performance results in TORCS of PROPEL agents initialized with neural policies obtained via DDPG, over 25 random seeds. Each entry is formatted as Lap-time / Crash-ratio, reporting median lap time in seconds over all the seeds (lower is better) and ratio of seeds that result in crashes (lower is better). A lap time of CR indicates the agent crashed and could not complete a lap for more than half the seeds.*

LENGTH	G-TRACK 3186M	E-ROAD 3260M	AALBORG 2588M	RUUDSKOGEN 3274M	ALPINE-2 3774M
IPPGPROGRAM-DDPG	97.76/.12	108.06/.08	140.48/.48	CR / 0.68	CR / 0.92
IPPGTREE-DDPG	78.47/0.16	85.46/.04	CR / 0.56	CR / 0.68	CR / 0.92

Classic Control

We present results from two classic control problems, Mountain-Car (with continuous actions) and Pendulum, in Table D.3. We use the OpenAI Gym implementations of these environments. More information about these environments can be found at the links: MountainCar and Pendulum.

In Mountain-Car the goal is to drive an under-powered car up the side of a mountain in as few time-steps as possible. In Pendulum, the goal is to swing a pendulum

Table D.2: *Generalization results in TORCS for IPPGTREE, where rows are training and columns are testing tracks. Each entry is formatted as IPPGPROGRAM / DDPG, and the number reported is the median lap time in seconds over all the seeds (lower is better). CR indicates the agent crashed and could not complete a lap for more than half the seeds.*

	G-TRACK	E-ROAD	AALBORG	RUUDSKOGEN	ALPINE-2
G-TRACK	-	95	CR	CR	CR
E-ROAD	84	-	CR	CR	CR
AALBORG	111	CR	-	CR	CR
RUUDSKOGEN	154	CR	CR	-	CR
ALPINE-2	CR	276	CR	CR	-

Table D.3: Performance results in Classic Control problems. Higher scores are better.

	MOUNTAINCAR	PENDULUM
PRIOR	00.59 ± 0.00	-875.53 ± 0.00
DDPG	97.16 ± 3.21	-132.70 ± 6.44
TRPO	93.03 ± 1.86	-131.54 ± 4.56
NDPS	66.98 ± 3.11	-435.71 ± 4.83
VIPER	64.86 ± 3.28	-394.11 ± 4.97
IPPGPROGRAM	95.63 ± 1.02	-187.71 ± 2.35
IPPGTREE	96.56 ± 2.81	-139.09 ± 3.31

up so that it stays upright. In both the environments an episode terminates after a maximum of 200 time-steps.

In Table D.3 we report the mean and standard deviation, over twenty-five random seeds, of the average scores over 100 episodes for the listed agents and environments. In Figure D.1 and Figure D.2 we show the improvements made over the prior by the IPPGPROGRAM agent in MountainCar and Pendulum respectively, with each iteration of the PROPEL algorithm.

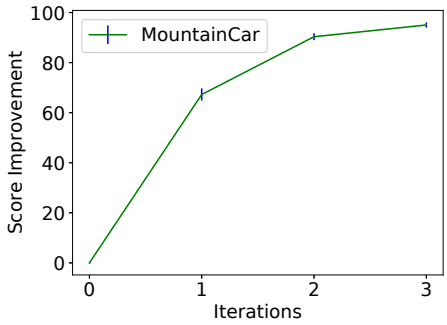


Figure D.1: Score improvements in the MountainCar environment over iterations of IPPGPROGRAM.

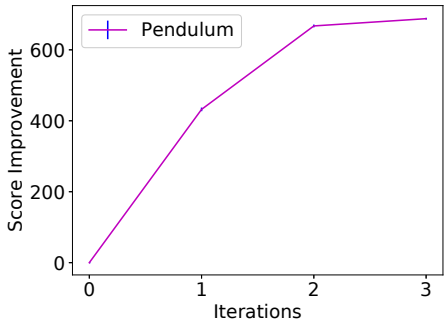


Figure D.2: Score improvements in the Pendulum environment over iterations of IPPGPROGRAM.

Appendix E

APPENDIX TO CHAPTER 6

E.1 Proofs for Chapter 6

Proof of Theorem 6.4.2. The first term TC^I_{FULL} should be obvious as the expert inspects the agent's overall behavior in each episode. Whenever something goes wrong in an episode, the expert labels the whole trajectory, incurring C^L_{FULL} each time. The remaining work is to bound the number of episodes where agent makes one or more mistakes. This quantity is bounded by the number of total mistakes made by the halving algorithm, which is at most the logarithm of the number of candidate functions (policies), $\log|\Pi_{\text{FULL}}| = \log\big(|\mathcal{M}||\Pi_{\text{LO}}|^{|\mathcal{G}|}\big) = \log|\mathcal{M}| + |\mathcal{G}|\log|\Pi_{\text{LO}}|$. This completes the proof. □

Proof of Theorem 6.4.1. Similar to the proof of Theorem 6.4.2, the first term TC^I_{FULL} is obvious. The second term corresponds to the situation where $\texttt{Inspect}_{\text{FULL}}$ finds issues. According to Algorithm 9, the expert then labels the subgoals and also inspects whether each subgoal is accomplished successfully, which incurs $C^L_{\text{HI}} + H_{\text{HI}}C^I_{\text{LO}}$ cost each time. The number of times that this situation happens is bounded by (a) the number of times that a wrong subgoal is chosen, plus (b) the number of times that all subgoals are good but at least one of the subpolicies fails to accomplish the subgoal. Situation (a) occurs at most $\log|\mathcal{M}|$ times. In situation (b), the subgoals chosen in the episode must come from $\mathcal{G}_{\text{opt}}$, and for each of these subgoals the halving algorithm makes at most $\log|\Pi_{\text{LO}}|$ mistakes. The last term corresponds to cost of $\texttt{Label}_{\text{LO}}$ operations. This only occurs when the meta-controller chooses a correct subgoal but the corresponding subpolicy fails. Similar to previous analysis, this situation occurs at most $\log|\Pi_{\text{LO}}|$ for each "good" subgoal ($g \in \mathcal{G}_{\text{opt}}$). This completes the proof. □

E.2 Additional Experimental Details

In our experiments, *success rate* and *external rewards* are reported as the trailing average over previous 100 episodes of training. For hierarchical imitation learning experiments in maze navigation domain, the success rate is only measured on separate test environments not used for training.

In addition to experimental results, in this section we describe our mechanism for

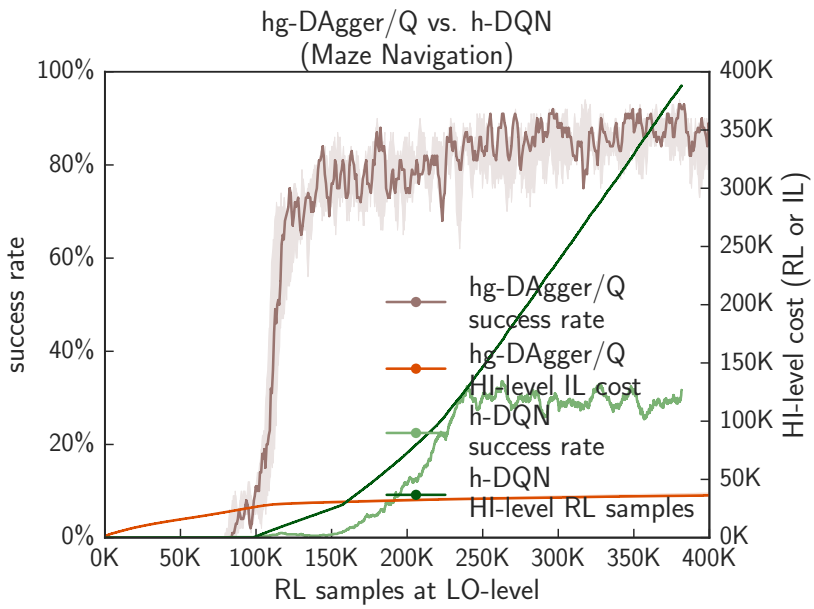


Figure E.1: *Maze navigation: hybrid IL-RL (full task) versus h-DQN (with 50% head-start).*

subgoal detection / terminal predicate for Montezuma's Revenge and how the Maze Navigation environments are created. Network architectures from our experiments are in Tables E.1 and E.2.

Maze Navigation Domain

We compare hg-DAgger/Q with the hierarchical reinforcement learning baseline (h-DQN, (Kulkarni et al., 2016)) with the same network architecture for the meta-controller and subpolicies as hg-DAgger/Q and similarly enhanced Q-learning procedure.

Similar to the Montezuma's Revenge domain, h-DQN does not work well for the maze domain. At the HI level, the planning horizon of 10–12 with 4–5 possible subgoals in each step is prohibitively difficult for the HI-level reinforcement learner and we were not able to achieve non-zero rewards within in any of our experiments. To make the comparison, we attempted to provide additional advantage to the h-DQN algorithm by giving it some head-start, so we ran h-DQN with 50% reduction in the horizon, by giving the hierarchical learner the optimal execution of the first half of the trajectory. The resulting success rate is in Figure E.1. Note that the hybrid IL-RL does not get the 50% advantage, but it still quickly outperforms h-DQN,

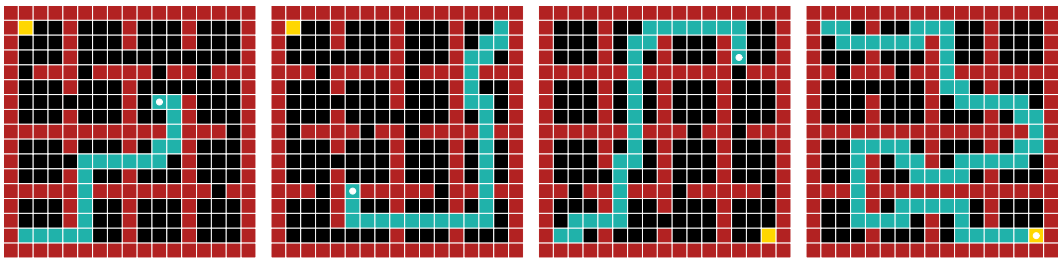


Figure E.2: *Maze navigation.* Sample random instances of the maze domain (different from main text). The 17 $\times$ 17 pixel representation of the maze is used as input for neural network policies.

which flattens out at 30% success rate.

Creating Maze Navigation Environments

We create 2000 maze navigation environments, 1000 of which are used for training and 1000 maps are used for testing. The comparison results for maze navigation (e.g., Figure 6.2) are all based on randomly selected environments among 1000 test maps. See Figure E.2 for additional examples of the environments created. For each map (environment instance), we start with a 17$\times$17 grid, which are divided into 4$\times$4 room structure. Initially, no door exists in between rooms. To create an instance of the maze navigation environment, the goal block (yellow) and the starting position are randomly selected (accepted as long as they are not the same). Next, we randomly select a wall separating two different room and replace a random red block (lava) along this wall with a door (black cell). This process continues until two conditions are satisfied:

- There is a feasible path between the starting location and the goal block (yellow)
- The minimum distance between start to goal is at least 40 steps. The optimal path can be constructed using graph search

Each of the 2000 environments create must satisfy both conditions. The expert labels for each environment come from optimal policy computed via value iteration (which is fast based on tabular representation of the given grid world).

Hyperparameters for Maze Navigation

The network architecture used for maze navigation is described in Table E.1. The only difference between subgoal policy networks and metacontroller network is the

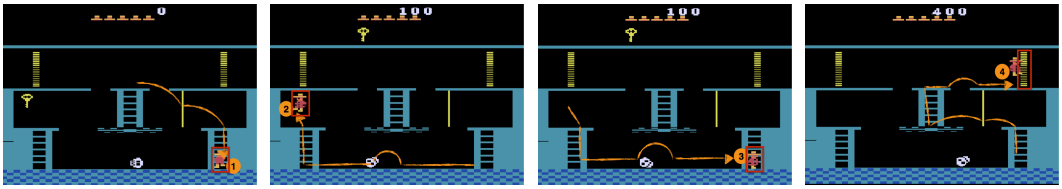


Figure E.3: *Montezuma's Revenge: Screenshots of the environment with 4 designated subgoals in sequence.*

number of output class (4 actions versus 5 subgoals). For our hierarchical imitation learning algorithms, we also maintain a small network along each subgoal policy for subgoal termination classification (one can also view the subgoal termination classifier as an extra head of the subgoal policy network).

The contextual input (state) to the policy networks consists of 3-channel pixel representation of the maze environment. We assign different (fixed) values to goal block, agent location, agent's trail and lava blocks. In our hierarchical imitation learning implementations, the base policy learner (DAgger and behavior cloning) update the policies every 100 steps using stochastic optimization. We use Adam optimizer and learning rate of 0.0005.

Table E.1: Network Architecture—Maze Domain

1: Convolutional Layer	32 filters, kernel size 3, stride 1
2: Convolutional Layer	32 filters, kernel size 3, stride 1
3: Max Pooling Layer	pool size 2
4: Convolutional Layer	64 filters, kernel size 3, stride 1
5: Convolutional Layer	64 filters, kernel size 3, stride 1
6: Max Pooling Layer	pool size 2
7: Fully Connected Layer	256 nodes, relu activation
8: Output Layer	softmax activation (dimension 4 for subpolicy, dimension 5 for meta-controller)

Montezuma's Revenge

Although the imitation learning component tends to be stable and consistent, the samples required by the reinforcement learners can vary between experiments with identical hyperparameters. In this section, we report additional results of our hybrid algorithm for the Montezuma's Revenge domain.

For the implementation of our hybrid algorithm on the game Montezuma's Revenge,

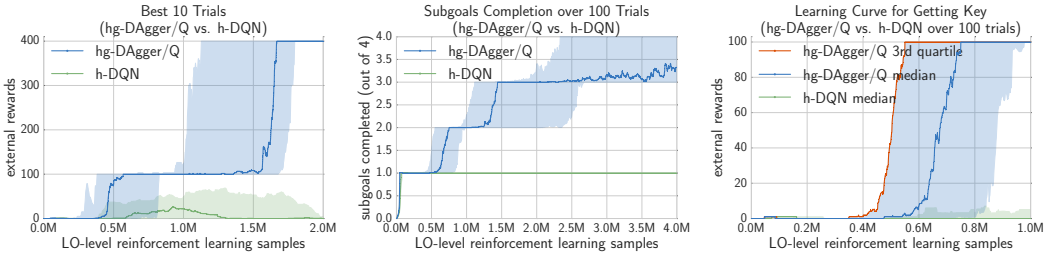


Figure E.4: *Montezuma's revenge: hybrid IL-RL versus hierarchical RL. (Left)* Median reward, min and max across the best 10 trials. The agent completes the first room in less than 2 million samples. The shaded region corresponds to min and max of the best 10 trials. *(Middle)* Median, first and third quartile of subgoal completion rate across 100 trials. The shaded region corresponds to first and third quartile. *(Right)* Median, first and third quartile of reward across 100 trials. The shaded region corresponds to first and third quartile. h-DQN only considers the first two subgoals to simplify the learning task.

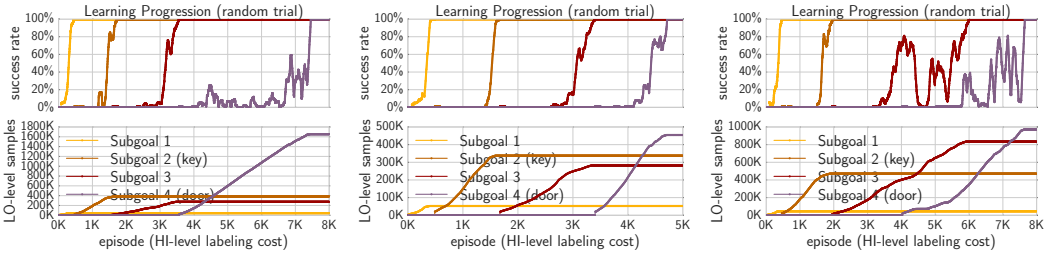


Figure E.5: *Montezuma's revenge: Learning progression of Algorithm 3 in solving the entire first room. The figures show three randomly selected successful trials.*

we decided to limit the computation to 4 million frames for the LO-level reinforcement learners (in aggregate across all 4 subpolicies). Out of 100 experiments, 81 out of 100 successfully learn the first 3 subpolicies, 89 out of 100 successfully learn the first 2 subpolicies. The last subgoal (going from the bottom of the stairs to open the door) proved to be the most difficult and almost half of our experiments did not manage to finish learning the fourth subpolicy within the 4 million frame limit (see Figure E.4 middle pane). The reason mainly has to do with the longer horizon of subgoal 4 compared to other three subgoals. Of course, this is a function of the design of subgoals and one can always try to shorten the horizon by introducing intermediate subgoals.

However, it is worth pointing out that even as we limit the h-DQN baseline to only 2 subgoals (up to getting the key), the h-DQN baseline generally tends to underperform our proposed hybrid algorithm by a large margin. Even with the given advantage we confer to our implementation of h-DQN, all of the h-DQN experiments failed to successfully master the second subgoal (getting the key). It is instructive to also

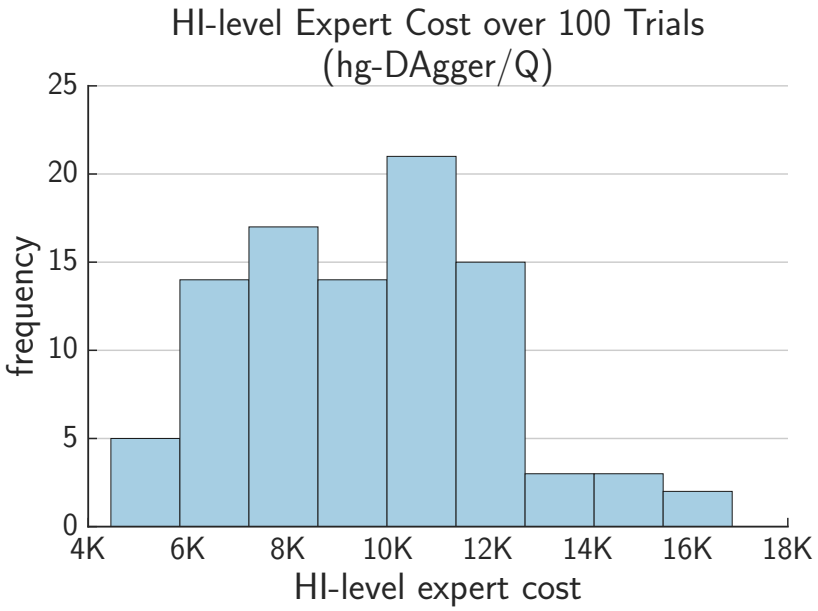


Figure E.6: *Montezuma's Revenge: Number of HI-level expert labels.* Distribution of Hi-level expert labels needed across 100 trials; the histogram excludes 6 outliers whose number of labels exceeds 20K for ease of visualization

examine the sample complexity associated with getting the key (the first positive external reward, see Figure E.4 right pane). Here the horizon is sufficiently short to appreciate the difference between having expert feedback at the HI level versus relying only on reinforcement learning to train the meta-controller.

The stark difference in learning performance (see Figure E.4 right) comes from the fact that the HI-level expert advice effectively prevents the LO-level reinforcement learners from accumulating bad experience, which is frequently the case for h-DQN. The potential corruption of experience replay buffer also implies at in our considered setting, learning with hierarchical DQN is no easier compared to flat DQN learning. Hierarchical DQN is thus susceptible to collapsing into the flat learning version.

Subgoal Detectors for Montezuma's Revenge

In principle, the system designer would select the hierarchical decomposition that is most convenient for giving feedback. For Montezuma's Revenge, we set up four subgoals and automatic detectors that make expert feedback trivial. The subgoals are landmarks that are described by small rectangles. For example, the door subgoal (subgoal 4) would be represented by a patch of pixel around the right door (see

Figure E.3 right). We can detect the correct termination / attainment of this subgoal by simply counting the number of pixels inside of the pre-specified box that has changed in value. Specifically in our case, subgoal completion is detected if at least 30% of pixels in the landmark's detector box changes.

Hyperparameters for Montezuma's Revenge

Table E.2: Network Architecture—Montezuma's Revenge

1: Conv. Layer	32 filters, kernel size 8, stride 4, relu
2: Conv. Layer	64 filters, kernel size 4, stride 2, relu
3: Conv. Layer	64 filters, kernel size 3, stride 1, relu
4: Fully Connected Layer	512 nodes, relu, normal initialization with std 0.01
5: Output Layer	linear (dimension 8 for subpolicy, dimension 4 for meta-controller)

Neural network architecture used is similar to (Kulkarni et al., 2016). One difference is that we train a separate neural network for each subgoal policy, instead of maintaining a subgoal encoding as part of the input into a policy neural network that shares representation for multiple subgoals jointly. Empirically, sharing representation across multiple subgoals causes the policy performance to degrade we move from one learned subgoal to the next (a phenomenon of catastrophic forgetting in deep learning literature). Maintaining each separate neural network for each subgoal ensures the performance to be stable across subgoal sequence. The metacontroller policy network also has similar architecture. The only difference is the number of output (4 output classes for metacontroller, versus 8 classes (actions) for each LO-level policy).

For training the LO-level policy with Q-learning, we use DDQN (Van Hasselt et al., 2016) with prioritized experience replay (Schaul et al., 2015b) (with prioritization exponent $\alpha = 0.6$, importance sampling exponent $\beta_0 = 0.4$). Similar to previous deep reinforcement learning work applied on Atari games, the contextual input (state) consists of four consecutive frames, each converted to grey scale and reduced to size 84×84 pixels. Frame skip parameter as part of the Arcade Learning Environment is set to the default value of 4. The repeated action probability is set to 0, thus the Atari environment is largely deterministic. The experience memory has capacity of 500K. The target network used in Q-learning is updated every 2000 steps. For stochastic

optimization, we use rmsProp with learning rate of 0.0001, with mini-batch size of 128.

E.3 Additional Related Work

Imitation Learning. Another dichotomy in imitation learning, as well as in reinforcement learning, is that of value-function learning versus policy learning. The former setting (Abbeel and Ng, 2004; Ziebart et al., 2008) assumes that the optimal (demonstrated) behavior is induced by maximizing an unknown value function. The goal then is to learn that value function, which imposes a certain structure onto the policy class. The latter setting (Daumé III et al., 2009; Ross et al., 2011b; Ho and Ermon, 2016) makes no such structural assumptions and aims to directly fit a policy whose decisions well imitate the demonstrations. This latter setting is typically more general but often suffers from higher sample complexity. Our approach is agnostic to this dichotomy and can accommodate both styles of learning. Some instantiations of our framework allow for deriving theoretical guarantees, which rely on the policy learning setting. Sample complexity comparison between imitation learning and reinforcement learning has not been studied much in the literature, perhaps with the exception of the recent analysis of AggreVaTeD (Sun et al., 2017).

Hierarchical Reinforcement Learning. Feudal RL is another hierarchical framework that is similar to how we decompose the task hierarchically (Dayan and Hinton, 1993; Dietterich, 2000; Vezhnevets et al., 2017). In particular, a feudal system has a manager (similar to our HI-level learner) and multiple submanagers (similar to our LO-level learners), and submanagers are given pseudo-rewards which define the subgoals. Prior work in feudal RL use reinforcement learning for both levels; this can require a large amount of data when one of the levels has a long planning horizon, which we demonstrate in our experiments. In contrast, we propose a more general framework where imitation learners can be used to substitute reinforcement learners to substantially speed up learning, whenever the right level of expert feedback is available. Hierarchical policy classes have been additional studied by (He et al., 2010), (Hausknecht and Stone, 2016), (Zheng et al., 2016), and (Andreas et al., 2017).

Learning with Weaker Feedback. Our work is motivated by efficient learning under weak expert feedback. When we only receive demonstration data at the high level, and must utilize reinforcement learning at the low level, then our setting can be viewed as an instance of learning under weak demonstration feedback. The

primary other way to elicit weaker demonstration feedback is with preference-based or gradient-based learning, studied by (Fürnkranz et al., 2012), (Loftin et al., 2016), and (Christiano et al., 2017).

Appendix F

APPENDIX TO CHAPTER 7

F.1 Variational Inference Derivation for Hidden Markov Models

In this section, we provide the mathematical derivation for the structured variational inference procedure. We focus on the training for Bayesian Hidden Markov Model, in particular the Forward-Backward procedure to complete the description of Algorithm 13. The mathematical details for other types of graphical models depend on the family of such models and should follow similar derivations. Further relevant details on stochastic variational inference can be found in (Hoffman et al., 2013; Johnson and Willsky, 2014; Beal, 2003). d

Settings. Given an arbitrarily ordered set of trajectories $U = \{U_1, \ldots, U_K, C\}$, let the coordination mechanism underlying each such U be governed by a true unknown model p, with global parameters θ. We suppress the agent/policy subscript and consider a generic featurized trajectory $x_t = [u_t, c_t] \ \forall t$. Let the latent role sequence for the same agent be $z = z_{1:T}$.

At any time t, each agent is acting according to a latent role $z_t \sim \texttt{Categorical}\{\bar{1}, \bar{2}, \ldots, \bar{K}\}$, which are the local parameters to the structured model.

Ideally, role and index asignment can be obtained by calculating the posterior $p(z|x, \theta)$, which is often intractable. One way to infer the role assignment is via approximating the intractable posterior $p(z|x, \theta)$ using Bayesian inference, typically via MCMC or mean-field variational methods. Since sampling-based MCMC methods are often slow, we instead aim to learn to approximate $p(z|x, \theta)$ by a simpler distribution q via Bayesian inference. In particular, we employ techniques from stochastic variational inference (Hoffman et al., 2013), which allows for efficient stochastic training on mini-batches that can naturally integrate with our imitation learning subroutine.

Structured Variational Inference for Unsupervised Role Learning. Consider a full probabilistic model:

$$p(\theta, z, x) = p(\theta) \prod_{t=1}^{T} p(z_t|\theta) p(x_t|z_t, \theta)$$

with global latent variables θ, local latent variables $z = \{z_t\}_{t=1}^T$. Posterior approximation is often cast as optimizing over a simpler model class $\mathcal{Q}$, via searching for global parameters θ and local latent variables z that maximize the evidence lower bound (ELBO) $\mathcal{L}$:

$$\begin{aligned}\log p(x) &\geqslant \mathbb{E}_q\left[\log p(z,\theta,x)\right] - \mathbb{E}_q\left[\log q(z,\theta)\right] \\ &\triangleq \mathcal{L}\left(q(z,\theta)\right).\end{aligned}$$

Maximizing $\mathcal{L}$ is equivalent to finding $q \in \mathcal{Q}$ to minimize the KL divergence $\mathrm{KL}\left(q(z,\theta|x)||p(z,\theta|x)\right)$.

For unsupervised structured prediction problem over a family of graphical model, we focus on the structured mean-field variational family, which factorizes q as $q(z,\theta) = q(z)q(\theta)$ (Hoffman and Blei, 2014) and decomposes the ELBO objective:

$$\begin{aligned}\mathcal{L} = \mathbb{E}_q[\log p(\theta] - \mathbb{E}_q[\log q(\theta] \\ + \mathbb{E}_q[\log(p(z,x|\theta)] - \mathbb{E}_q[\log(q(z))].\end{aligned} \tag{F.1}$$

This factorization breaks the dependency between θ and z, but not between single latent states z_t, unlike variational inference for i.i.d data (Kingma and Welling, 2013).

Variational inference optimizes the objective $\mathcal{L}$ typically using natural gradient ascent over global factors $q(\theta)$ and local factors $q(z)$. (Under mean-field assumption, optimization typically proceeds via alternating updates of θ and z.) Stochastic variational inference performs such updates efficiently in mini-batches. For graphical models, structured stochastic variational inference optimizes $\mathcal{L}$ using *natural gradient* ascent over global factors $q(\theta)$ and message-passing scheme over local factors $q(z)$. We assume the prior $p(\theta)$ and complete conditionals $p(z_t, x_t|\theta)$ are conjugate pairs of exponential family, which gives natural gradient of $\mathcal{L}$ with respect to $q(\theta)$ convenient forms (Johnson and Willsky, 2014). Denote the exponential family forms of $p(\theta)$ and $p(z_t, y_t|\theta)$ by:

$$\begin{aligned}\ln p(\theta) &= \langle \eta_\theta, t_\theta(\theta)\rangle - A_\theta(\eta_\theta) \\ \ln p(z_t, x_t|\theta) &= \langle \eta_{zx}(\theta), t_{zx}(z_t, x_t)\rangle - A_{zx}(\eta_{zx}(\theta))\end{aligned}$$

where η_θ and η_{zx} are functions indicating natural parameters, t_θ and t_{zx} are sufficient statistics and $A(\cdot)$ are log-normalizers ((Blei et al., 2017)). Note that in general, different subscripts corresponding to η, t, A indicate different function parameterization (not simply a change in variable value assignment). Conjugacy in

the exponential family yields that (Blei et al., 2017):

$$t_\theta(\theta) = [\eta_{zx}(\theta), -A_{zx}(\eta_{zx}(\theta))]$$

and that

$$p(\theta|z_t, x_t) \propto \exp\{\langle \eta_\theta + [t_{zx}(z_t, x_t), 1], t_\theta(\theta)\rangle\} \tag{F.2}$$

Conjugacy in the exponential family also implies that the optimal $q(\theta)$ is in the same family (Blei et al., 2017), i.e.

$$q(\theta) = \exp\{\langle \widetilde{\eta}_\theta, t_\theta(\theta)\rangle - A_\theta(\widetilde{\eta}_\theta)\}$$

for some natural parameters $\widetilde{\eta}_\theta$ of $q(\theta)$.

To optimize over global parameters $q(\theta)$, conjugacy in the exponential family allows obtaining convenient expression for the gradient of $\mathcal{L}$ with respect to natural parameters $\widetilde{\eta}_\theta$. The derivation is shown similarly to (Johnson and Willsky, 2014) and (Blei et al., 2017) - we use simplified notations $\widetilde{\eta} \triangleq \widetilde{\eta}_\theta, \eta \triangleq \eta_\theta, A \triangleq A_\theta$, and $t(z,x) \triangleq \sum_{t=1}^{T} [t_{zx}(z_t, x_t), 1]$. Taking advantage of the exponential family identity $\mathbb{E}_{q(\theta)}[t_\theta(\theta)] = \nabla A(\widetilde{\eta})$, the objective $\mathcal{L}$ can be re-written as:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{q(\theta)q(z)}\left[\ln p(\theta|z,x) - \ln q(\theta)\right] \\ &= \langle \eta + \mathbb{E}_{q(z)}[t(z,x)], \nabla A(\widetilde{\eta})\rangle - (\langle \widetilde{\eta}, \nabla A(\widetilde{\eta})\rangle - A(\widetilde{\eta}))\end{aligned}$$

Differentiating with respect to $\widetilde{\eta}$, we have that

$$\nabla_{\widetilde{\eta}}\mathcal{L} = \left(\nabla^2 A(\widetilde{\eta})\right)\left(\eta + \mathbb{E}_{q(z)}[t(z,x)] - \widetilde{\eta}\right)$$

The *natural gradient* of $\mathcal{L}$, denoted $\widetilde{\nabla}_{\widetilde{\eta}}$, is defined as $\widetilde{\nabla}_{\widetilde{\eta}} \triangleq \left(\nabla^2 A(\widetilde{\eta})\right)^{-1} \nabla_{\widetilde{\eta}}$. And so the natural gradient of $\mathcal{L}$ can be compactly described as:

$$\widetilde{\nabla}_{\widetilde{\eta}}\mathcal{L} = \eta + \sum_{t=1}^{T} \mathbb{E}_{q(z_t)}\{[t_{zx}(z_t, x_t), 1]\} - \widetilde{\eta} \tag{F.3}$$

Thus a stochastic natural descent update on the global parameters $\widetilde{\eta}_\theta$ proceeds at step n by sampling a mini-batch x_t and taking the global update with step size ρ_n:

$$\widetilde{\eta}_\theta \leftarrow (1-\rho_n)\widetilde{\eta}_\theta + \rho_n(\eta_\theta + b^\top \mathbb{E}_{q^*(z_t)}[t(z_t, x_t)]) \tag{F.4}$$

where b is a vector of scaling factors adjusting for the relative size of the mini-batches. Here the global update assumes optimal local update $q^*(z)$ has been computed. In each step however, the local factors $q^*(z_t)$ are computed with mean

field updates and the current value of $q(\theta)$ (analogous to coordinate ascent). In what follows, we provide the derivation for the update rules for Hidden Markov Models, which are the particular instantiation of the graphical model we use to represent the role transition for our multi-agent settings.

Variational factor updates via message passing for Hidden Markov Models. For HMMs, we can view global parameters θ as the parameters of the underlying HMMs such as transition matrix and emission probabilities, while local parameters z govern hidden state assignment at each time step.

Fixing the global parameters, the local updates are based on message passing over the graphical model. The exact mathematical derivation depends on the specific graph structure. The simplest scenario is to assume independence among z_t's, which resembles naive Bayes. We instead focus on Hidden Markov Models to capture first-order dependencies in role transitions over play sequences. In this case, global parameters $\theta = (p_0, P, \phi)$ where $P = \left[P_{ij}\right]_{i,j=1}^{K}$ is the transition matrix with $P_{ij} = p(z_t = j | z_{t-1} = i)$, $\phi = \{\phi_i\}_{i=1}^{K}$ are the emission parameters, and p_0 is the initial distribution.

Consider a Bayesian HMM on K latent states. Priors on the model parameters include the initial state distribution p_0, transition matrix P with rows denoted $p_1, \ldots, p_K$, and the emission parameters $\phi = \{\phi_i\}_{i=1}^{K}$. In this case we have the global parameters $\theta = (p_0, P, \phi)$. For Hidden Markov Model with observation $x_{1:T}$ and latent sequence $z_{1:T}$, the generative model over the parameters is given by $\phi_i \sim p(\phi)$ (i.i.d from prior), $p_i \sim \text{Dir}(\alpha_i)$, $z_1 \sim p_0$, $z_{t+1} \sim p_{z_t}$, and $x_t \sim p(x_t | \phi_{z_t})$ (conditional distribution given parameters ϕ). We can also write the transition matrix:

$$P = \begin{bmatrix} p_1 \\ \vdots \\ p_K \end{bmatrix}$$

The Bayesian hierarchical model over the parameters, hidden state sequence $z_{1:T}$, and observation sequence $y_{1:T}$ is

$$\phi_i \overset{\text{iid}}{\sim} p(\phi), p_i \sim \text{Dir}(\alpha_i)$$
$$z_1 \sim p_0, z_{t+1} \sim p_{z_t}, x_t \sim p(x_t | \phi_{z_t})$$

For HMMs, we have a full probabilistic model: $p(z, x | \theta) = p_0(z_1) \prod_{t=1}^{T} p(z_t | z_{t-1}, P) p(x_t | z_t, \phi)$. Define the likelihood potential $L_{t,i} = p(x_t | \phi_i)$, the likelihood of the latent sequence,

given observation and model parameters, is as follows:

$$p(z_{1:T}|x_{1:T}, P, \phi) = \exp\left(\log p_0(z_1) + \sum_{t=2}^{T} \log P_{z_{t-1},z_t} + \sum_{t=1}^{T} \log L_{t,z_t} - Z\right) \tag{F.5}$$

where Z is the normalizing constant. Following the notation and derivation from (Johnson and Willsky, 2014), we denote $p(z_{1:T|x_{1:T},P,\phi}) = \text{HMM}(p_0, P, L)$. Under mean field assumption, we approximate the true posterior $p(P, \phi, z_{1:T}|x_{1:T})$ with a mean field variational family $q(P)q(\phi)q(z_{1:T})$ and update each variational factor in turn while fixing the others.

Fixing the global parameters θ, taking expectation of log of (F.5), we derive the update rule for $q(z)$ as $q(z_{1:T}) = \text{HMM}(\widetilde{P}, \widetilde{p}_0, \widetilde{L})$ where:

$$\begin{aligned}
\widetilde{P}_{j,k} &= \exp\{\mathbb{E}_{q(P)} \ln(P_{j,k})\} \\
\widetilde{p}_{0,k} &= \exp\{\ln \mathbb{E}_{q(p_0)} p_{0,k}\} \\
\widetilde{L}_{t,k} &= \exp\{\mathbb{E}_{q(\phi_k)} \ln(p(x_t|z_t = k))\}
\end{aligned}$$

To calculate the expectation with respect to $q(z_{1:T})$, which is necessary for updating other factors, the `Forward-Backward` recursion of HMMs is defined by forward messages F and backward messages B:

$$F_{t,i} = \sum_{j=1}^{K} F_{t-1,j} \widetilde{P}_{j,i} \widetilde{L}_{t,i} \tag{F.6}$$

$$B_{t,i} = \sum_{j=1}^{K} \widetilde{P}_{i,j} \widetilde{L}_{t+1,j} B_{t+1,j} \tag{F.7}$$

$$\begin{aligned}
F_{1,i} &= p_0(i) \\
B_{T,i} &= 1
\end{aligned}$$

As a summary, calculating the gradient w.r.t z yields the following optimal variational distribution over the latent sequence:

$$\begin{aligned}
q^*(z) \propto \exp\Big(&\mathbb{E}_{q(P)}[\ln p_0(z_1)] + \sum_{t=2}^{T} \mathbb{E}_{q(P)}[\log P_{z_{t-1},z_t}] \\
&+ \sum_{t=1}^{T} \mathbb{E}_{q(\phi)} \ln[p(x_t|z_t)]\Big),
\end{aligned} \tag{F.8}$$

which gives the local updates for $q^*(z)$, given current estimates of P and ϕ:

$$\widetilde{P}_{j,k} = \exp\left[\mathbb{E}_{q(P)} \ln(P_{j,k})\right] \tag{F.9}$$

$$\widetilde{p}(x_t|z_t = k) = \exp\left[E_{q(\phi)} \ln p(x_t|x_t = k)\right], \tag{F.10}$$

for $k = 1, \ldots, K$, $t = 1, \ldots, T$, and then use $p_0, \widetilde{P}, \widetilde{p}$ to run the forward-backward algorithm to compute the update $q^*(z_t = k)$ and $q^*(z_{t-1} = j, z_t = k)$. The forward-backward algorithm in the local update step takes $O(K^2T)$ time for a chain of length T and K hidden states.

Training to learn model parameters for HMMs. Combining *natural gradient* step with message-passing scheme for HMMs yield specific update rules for learning the model parameters. Again for HMMs, the global parameters are $\theta = (p_0, P, \phi)$ and local variables $z = z_{1:T}$. Assuming the priors on observation parameter $p(\phi_i)$ and likelihoods $p(x_t|\phi_i)$ are conjugate pairs of exponential family distribution for all i, the conditionals $p(\phi_i|x)$ have the form as seen from equation F.2:

$$p(\phi_i|x) \propto \exp\{\langle \eta_{\phi_i} + [t_{x,i}(x), 1], t_{\phi_i}(\phi_i)\rangle\}$$

For structured mean field inference, the approximation $q(\theta)$ factorizes as $q(P)q(p_0)q(\phi)$. At each iteration, stochastic variational inference sample a sequence $x_{1:T}$ from the data set (e.g. trajectory from any randomly sampled player) and perform stochastic gradient step on $q(P)q(p_0)q(\phi)$. In order to compute the gradient, we need to calculate expected sufficient statistics w.r.t the optimal factor for $q(z_{1:T})$, which in turns depends on current value of $q(P)q(p_0)q(\phi)$.

Following the notation from (Johnson and Willsky, 2014), we write the prior and mean field factors as

$$p(p_i) = \mathrm{Dir}(\alpha_i), p(\phi_i) \propto \exp\{\langle \eta_{\phi_i}, t_{\phi_i}(\phi_i)\rangle\}$$
$$q(p_i) = \mathrm{Dir}(\widetilde{\alpha}_i), q(\phi_i) \propto \exp\{\langle \widetilde{\eta}_{\phi_i}, t_{\phi_i}(\phi_i)\rangle\}$$

Using message passing scheme as per equations (F.6) and (F.7), we define the

intermediate quantities:

$$\widehat{t}_{x,i} \triangleq \mathbb{E}_{q(z_{1:T})} \sum_{t=1}^{T} \mathbb{I}[z_t = i] t_{x,i}(x_t)$$

$$= \sum_{t=1}^{T} F_{t,i} B_{t,i} [t_{x,i}(x_t), 1] / Z \tag{F.11}$$

$$(\widehat{t}_{trans,i})_j \triangleq \mathbb{E}_{q(z_{1:T})} \sum_{t=1}^{T-1} \mathbb{I}[z_t = i, z_{t+1} = j]$$

$$= \sum_{t=1}^{T-1} F_{t,i} \widetilde{P}_{i,j} \widetilde{L}_{t+1,j} B_{t+1,j} / Z \tag{F.12}$$

$$(\widehat{t}_{init})_i \triangleq \mathbb{E}_{q(z_{1:T})} \mathbb{I}[z_1 = i] = \widetilde{p}_0 B_{1,i} / Z \tag{F.13}$$

where $Z \triangleq \sum_{i=1}^{K} F_{T,i}$ is the normalizing constant, and $\mathbb{I}$ is the indicator function.

Given these expected sufficient statistics, the specific update rules corresponding to the natural gradient step in the natural parameters of $q(P)$, $q(p_0)$, and $q(\phi)$ become:

$$\widetilde{\eta}_{\phi,i} \leftarrow (1-\rho)\widetilde{\eta}_{\phi,i} + \rho(\eta_{\phi,i} + b^\top \widehat{t}_{x,i}) \tag{F.14}$$

$$\widetilde{\alpha}_i \leftarrow (1-\rho)\widetilde{\alpha}_i + \rho(\alpha_i + b^\top \widehat{t}_{trans,i}) \tag{F.15}$$

$$\widetilde{\alpha}_0 \leftarrow (1-\rho)\widetilde{\alpha}_0 + \rho(\alpha_0 + b^\top \widehat{t}_{init,i}) \tag{F.16}$$

Algorithm 20 Coordinated Structure Learning: `LearnStructure` $\{U_1, \ldots, U_K, C, \theta, \rho\} \mapsto q(\theta, z)$

Input: Set of trajectories $U = \{U_k\}_{k=1}^K$. Context C
Previous parameters $\theta = (p_0, \theta^P, \theta^\phi)$, stepsize ρ

1: $X_k = \{x_{t,k}\}_{t=1}^T = \{[u_{t,k}, c_t]\}$ $\forall t, k . X = \{X_k\}_{k=1}^K$
2: Local update: Compute $\widetilde{P}$ and $\widetilde{p}$ per equation F.9 and F.10
and compute $q(z) =$ `Forward-Backward`$(X, \widetilde{P}, \widetilde{p})$
3: Global update of θ, per equations F.14, F.15, and F.16.

output Updated model $q(\theta, z) = q(\theta)q(z)$

F.2 Experimental Evaluation

Batch-Version of Algorithm 12 for Predator-Prey

Visualizing Role Assignment for Soccer

The Gaussian components of latent structure in figure 7.7 give interesting insight about the latent structure of the demonstration data, which correspond to a popular formation arrangement in professional soccer. Unlike the predator-prey domain,

Algorithm 21 Multi-Agent Data Aggregation Imitation Learning: $\texttt{Learn}(A_1, A_2, \ldots, A_K, C|D)$

Input: Ordered actions $A_k = \{a_{t,k}\}_{t=1}^{T} \;\; \forall k$, context $\{c_t\}_{t=1}^{T}$
Input: Aggregating data set $D_1, .., D_K$ for each policy
Input: base routine $\texttt{Train}(S, A)$ mapping state to actions

1: **for** $t = 0, 1, 2, \ldots, T$ **do**
2: Roll-out $\hat{a}_{t+1,k} = \pi_k(\hat{s}_{t,k}) \quad \forall$ agent k
3: Cross-update for each policy $k \in \{1, \ldots, K\}$
$\hat{s}_{t+1,k} = \varphi_k\left([\hat{a}_{t+1,1}, \ldots, \hat{a}_{t+1,k}, \ldots, \hat{a}_{t+1,K}, c_{t+1}]\right)$
4: Collect expert action $a^*_{t+1,k}$ given state $\hat{s}_{t+1,k} \;\; \forall k$
5: Aggregate data set $D_k = D_k \cup \{\hat{s}_{t+1,k}, a^*_{t+1,k}\}_{t=0}^{T-1}$
6: **end for**
7: $\pi_k \leftarrow \texttt{Train}(D_k)$

output K new policies $\pi_1, \pi_2, \ldots, \pi_K$

however, the players are sometimes expected to switch and swap roles. Figure F.1 displays the tendency that each learning policy k would takes on other roles outside of its dominant mode. Policies indexed $0-3$ tend to stay most consistent with the

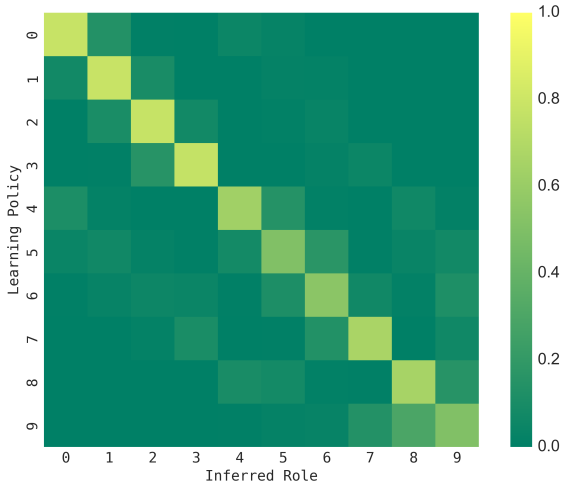


Figure F.1: *Role frequency assigned to policy, according to the maximum likelihood estimate of the latent structured model*

prescribed latent roles. We observe that these also correspond to players with the least variance in their action trajectories. Imitation loss is generally higher for less consistent roles (e.g. policies indexed $8-9$). Intuitively, entropy regularization encourages a decomposition of roles that result in learning policies as decoupled as possible, in order to minimize the imitation loss.