Towards Generalization and Efficiency of Reinforcement Learning

Wen Sun

March 8th, 2018

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
J. Andrew Bagnell, Chair
Geoffrey J. Gordon,
Martial Hebert,
Bryon Boots, Georgia Institute of Technology

Thesis proposal submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

©Wen Sun, 2018
Abstract

In classic supervised machine learning, a learning agent behaves as a passive observer: it receives examples from some external environment which it has no control over and then makes predictions. The predictions the agent made will not affect any future examples it will see (i.e., examples are identically and independently sampled from some unknown distribution). That is, the agent will not actively intervene the environment. Reinforcement Learning, on the other hand, is fundamentally interactive: an autonomous agent must learn how to behave in an unknown, uncertain, and possibly hostile environment, by actively interacting with the environment to collect useful feedback to improve its sequential decision making ability. The RL agent will also intervene in the environment: the agent makes decisions which in turn affects further evolution of the environment.

Because of its generality—most machine learning problems can be viewed as special cases—RL is hard. As there is no direct supervision, one central challenge in RL is how to explore an unknown environment and collect useful feedback efficiently. In recent RL success stories (e.g., super-human performance on video games [Mnih et al., 2015]), we notice that most of them rely on random exploration strategies, such as $\epsilon$-greedy. While $\epsilon$-greedy is simple and asymptotically optimal, it requires large number of interactions with the environment before it can learn anything useful. Similarly, policy gradient method such as REINFORCE [Williams, 1992], perform exploration by injecting randomness into action space and hope the randomness can lead to a good sequence of actions that achieves high total reward. The theoretical RL literature has developed more sophisticated algorithms for efficient exploration (e.g., [Azar et al., 2017]), however, the sample complexity of these near-optimal algorithms has to scale exponentially with respect to key parameters of underlying systems such as dimensions of state and action space. Such exponential dependence prohibits a direct application of these theoretically elegant RL algorithms to large-scale applications. In summary, without any further assumptions, RL is hard, both in practice and in theory.

In this work, we attempt to gain purchase on the RL problem by introducing additional assumptions and sources of information, and then reducing the resulting RL problem to simpler problems which we understand well and know how to solve. The first contribution of this work comes from a reduction of policy evaluation to no-regret online learning. As no-regret online learning is an active research area that has well-established theoretical foundation and appealing practical usage, such a reduction creates a new family of algorithms for provably correct policy evaluation under very weak assumptions on the generating process. This enables us to have correct bootstrapping policy evaluation algorithms without the requirement of a Markov process. Further the reduction allows any new, faster no-regret online algorithm to immediately translate to a faster policy evaluation algorithm. The second contribution of this work comes from improving RL sample efficiency via Imitation Learning (IL). Imitation Learning reduces policy improvement to classic supervised learning, which is a well-established research area and has provably correct algorithms along with efficient implementations. We study in both theory and in practice how one can imitate experts to reduce sample complexity compared to a pure RL approach. The third contribution of this work comes from leveraging efficient model-base optimal control by using a reduction of RL to IL. We explore the possibilities of learning local models and then using model-based optimal control solvers to compute an intermediate “expert” for efficient policy improvement via imitation. We propose a general framework, named Dual Policy Iteration (DPI), which maintains two policies, an apprentice policy and an expert policy, and alternatively updates the apprentice policy via imitating the expert policy while updates the expert policy via
model-based optimal control with a learned local model. Furthermore, we show a general convergence analysis that extends the existing approximate policy iteration theories to DPI. DPI generalizes and provides the first theoretical foundation for recent successful practical RL algorithms such as ExIt and AlphaGo Zero [Anthony et al., 2017; Silver et al., 2017], and provides a theoretical sound and practically efficient way of unifying model-based and model-free RL approaches.
## Contents

1 Introduction .......................... 1

2 Background ........................ 5
   2.1 Online Learning ...................... 5
   2.2 Markov Decision Process .......... 6

3 Online Bellman Residual and Temporal Difference Algorithms ....... 9
   3.1 Introduction ......................... 9
   3.2 Preliminaries ......................... 10
   3.3 Online Bellman Residual Algorithms .. 11
   3.4 Online TD Learning via Online Mirror Descent and Implicit Online Learning .... 12
   3.5 Experiment .......................... 14

4 Efficient Reinforcement Learning via Imitation .................. 16
   4.1 Introduction ......................... 16
   4.2 Regret Analysis for IL with Perfect Oracle ....... 18
   4.3 Differentiable Imitation Learning ....... 20
   4.4 Sample-based Practical Algorithms .... 22
   4.5 Experiment .......................... 24

5 Generalizing Policy Iteration via Dual Policy Iteration ....... 28
   5.1 Introduction ......................... 28
   5.2 Preliminaries ......................... 29
   5.3 Dual Policy Iteration .............. 30
   5.4 Unifying Model-free RL and Model-based RL: A Practical Algorithm .... 34
   5.5 Application to Robust Policy Optimization .... 37
   5.6 Experiment .......................... 37

6 Proposed Work ........................ 41
   6.1 Proposed Work One: Temporal Difference Learning via Dynamic Programming .... 41
   6.2 Proposed Work Two: Advantage Weighted Model-Free Imitation Learning .... 44
   6.3 Proposed Work Three: No-Regret Approximate Policy Iteration ......... 49

7 Conclusion .......................... 51
Chapter 1

Introduction

A fundamental challenge in Artificial Intelligence (AI), robotics, and language processing is sequential prediction: to reason, plan, and make a sequence of predictions or decisions to minimize accumulated cost, achieve a long-term goal, or optimize for a loss acquired only after many predictions. Reinforcement Learning (RL), as a general framework for learning from experience to make predictions and decisions, is often considered as one of the perfect tools for solving such a challenge in AI. Recently, equipped with the advancement from Deep Learning literature, we have dramatically advanced the state-of-the-art of RL on a number of applications including simulated high-dimensional robotics control, video games, and board games [Mnih et al., 2015, Schulman et al., 2015a, Silver et al., 2017, 2016].

Classic supervised machine learning considers settings where training examples come all at once and a learner makes predictions that have not effect on future examples it will receive, i.e., the examples the learner received are generated from a fixed unknown distribution, identically and independently, without being intervened by the learner. Reinforcement Learning (RL), on the other hand, is interactive, where an autonomous agent must learn how to behave in an unknown, uncertain, and possibly hostile environment. By interacting with the environment, the agent receives sensory feedback that describes the current state of the environment, and reward signals that correspond to achieving some specific task. There is no human in the loop that tries to provide direct supervision to the agent. The goal in RL is to learn an optimal strategy (i.e., a policy) that maximizes the total reward.

Because of its generality—RL is a powerful framework that summarizes many special machine learning algorithms and applications—RL is hard. Without any prior knowledge about the environment where an RL agent is operating, the RL agent needs to explore: it must explore unknown part of the environment or try actions that have not been tested before to discover if there are high-reward regions or better behavior strategies. The exploration strategies an RL agent leverages largely determine the sample complexity: the amount of interaction with the environment before the agent can learn a near-optimal policy. Most of the existing successful practical RL algorithms typical deploy the simplest ever exploration strategy—random exploration. For instance, in Q-learning, we often explore by \( \epsilon \)-greedy: with probability \( \epsilon \), we randomly pick an action; otherwise we follow suggestions from the latest Q function. For non-zero \( \epsilon \), while \( \epsilon \)-greedy guarantees asymptotic optimality (i.e., with infinite amount of training time and interaction with the environment, we will discover the optimal policy), it requires huge amount of interactions with the environment. Similarly, policy gradient based method such as REINFORCE [Williams, 1992], relies on
stochastic policy to introduce randomness in the action space, such that by randomly trying different actions, we hope we can discover a sequence of behaviors that lead to higher total reward, and then adapt the policy in a way such that the likelihood of generating such a good sequence of behaviors is increased. Such purely randomized exploration strategies often can work when one does not care the cost of samples (e.g., simulation), but inevitably leads to poor sample efficiency. This is one of the reasons why today’s most successful RL stories are still living in ideal playgrounds: Atari video games, board games, and robotics simulations.

RL is even hard in simplified worlds (e.g., discrete Markov Decision Processes). Looking into the theoretical RL literature where researchers have been focusing on developing provably sample efficient RL algorithms for discrete MDPs, we notice that in the worst case, the sample complexity of any RL algorithms, have to suffer an exponential dependency on the key parameters of the underlying MDPs such as dimensions of state and action spaces. Such exponential dependency potentially will prohibits a direct application of these mathematically beautiful RL algorithms on real-world large-scale problems. What is even worse is that such exponential dependency is not improvable, i.e., there exists a set of MDPs where no RL algorithm can break such dependency!

Hence, in general, without any further assumptions, RL is hard. Our main focus in this thesis is to improve the efficiency of RL, by introducing additional assumptions and sources of information, and then reducing RL to simpler problems which we know how to solve optimally both in theory and in practice. We proceed in three directions:

1. We generalize policy evaluation [Baird, 1995, Sutton, 1988b] by providing a reduction from policy evaluation to no-regret online learning. As no-regret online learning is an extremely active research area, such reduction creates a family of new algorithms for provably correct policy evaluation under very weak assumptions on the generating process. Also any new, faster no-regret online learning algorithm can be immediately plugged into the reduction framework and leads to a new, faster policy evaluation algorithm. As policy evaluation plays an important role in almost all RL algorithms (e.g., actor-critic [Konda and Tsitsiklis, 2000]), a faster and more general policy evaluation approach can directly benefit these RL algorithms. We detail our contributions on policy evaluation in Chapter 3.

2. We improve sample efficiency of RL via imitating an expert. We consider the imitation learning setting where we have access to expert during training (but not in test time) and also the reward signals during training [Chang et al., 2015b, Choudhury et al., 2017a, Ross and Bagnell, 2014, Sun et al., 2017]. With the expert supervision, we reduce policy iteration to supervised learning. Though with reward signals, one may wonder why we need Imitation Learning while we can just perform RL. We argue that by leveraging an expert and the reward signals—combining reinforcement and imitation, not only we can learn a good policy that is in par with the expert (hence less sensitive to local optimality) with up to exponentially less samples than any RL approaches (hence more sample efficient), we can also learn a policy that outperforms the expert, with reasonable assumptions. We provide detailed analysis of IL versus RL in sample complexity, and a set of practical IL algorithms that can leverage modern deep networks in Chapter 4.

3. We improve sample efficiency of RL via model-based optimal control by using a reduction of RL to IL. We propose a new policy iteration scheme, Dual Policy Iteration...
(DPI)—a general framework that provides a principle way of unifying model-free RL and model-based RL approaches. Our DPI provides a framework where we maintain and alternatively update two policies: a fast reactive policy that aims to provide generalization, and locally optimal policies that are computed by model-based optimal control approaches with learned local dynamics. We can use any stochastic optimal control algorithms (e.g., Sun et al. [2016b], Todorov and Li [2005]) to compute policies that are locally optimal around the current reactive, non-linear policy. The reactive policy then treats the locally optimal policy as an intermediate “expert” and leverage the expert’s direct supervision to improve itself (i.e., policy iteration is reduced to simple supervised learning). We present the detailed convergence analysis and a practical algorithm in Chapter 5.

Our work also opens quite a few new problems, which we propose as future work in this document (Chapter 6):

1. In Chapter 3 we successfully extended Bellman Residual minimization based algorithms to a much broader setting: any no-regret and stable online algorithms can be used to minimize Bellman Residual, with arbitrary function approximations, to guarantee small prediction errors. Though in Chapter 3 we generalize TD learning such that we can apply mirror descent based algorithms on TD loss, we conjecture that similar to Bellman Residual, we can apply any no-regret online learner, including second order no-regret learner (e.g., Online Newton Step [Hazan et al. 2007]), on TD loss to achieve small prediction error. We propose to first focus on the simpler setting where the underlying sample generating process is a Markov chain. Instead of learning a single predictor, we maintain and learn a finite number of predictors, which are updated via Dynamic Programming in a backward fashion: we train the last predictor, and then the second last predictor with information bootstrapped from the last predictor, and so on, till the first predictor. We believe that analysis from PSDP [Bagnell et al. 2004] can be leveraged here to prove the correctness of this DP-based TD learning approach, which we name as TD-DP (Temporal Difference Learning via Dynamic Programming).

2. In Chapter 4 we showed how we can efficiently leverage an expert oracle during training to dramatically reduce sample complexity and speed up learning. One limitation of the work presented in Chapter 4 is that we need experts during the training loop. A natural question is that what if the expert is not available anymore during training? In this proposed work, we consider the classic apprenticeship learning setting where we only have a set of demonstrations provided by the expert before the learning process. We discuss previous works designed for this setting and show that in worst case, all these methods, including reduction to classic supervised learning [Syed 2010], margin based learning [Ratliff et al. 2006], and a game-theoretical formulation [Syed 2010], have the same sample complexity that scales like $T^2\epsilon$, where $T$ is the planning horizon and $\epsilon$ indicates how far away the performance of the learned policy is to the expert. To break such low bound, we propose to introduce two assumptions: (1) we can reset the system and (2) we can roll out a policy on the system from any state and receive the accumulative reward at the end of the roll-out process. We propose a new algorithm that leverages no-regret online learning, and prove that it outperforms all previous methods (e.g., [Ratliff et al. 2006], [Syed 2010]) in sample complexity. In practice, we plan to implement the new algorithm and to see if the better theoretical guarantee also leads to a better experimental performance in practice.
3. We plan to explore using no-regret update for policy improvement. During approximate policy iteration, in order to make sure the new policy outperforms the old one, the classic strategy is to force the new policy to stay “close” to the old one [Bagnell and Schneider, 2003, Bagnell et al., 2004, Kakade, 2002, Kakade and Langford, 2002, Scherrer, 2014]. However, the amount of improvement one can achieve usually linear shrinks with respect to the closeness to the old policy, which potentially makes the policy improvement very small in practice. However, from the IL literature, we known how to provably compute a policy that outperforms the expert policy (if the expert policy is in the policy class) by one-step deviation (e.g., AggreVaTe [Ross and Bagnell, 2014] provides such a guarantee). Hence, we propose a new approximate policy iteration algorithm, where in each iteration, we treat the current policy as the expert policy, and invoke IL algorithm, i.e., AggreVaTe, to compute a new policy that can provably achieve one-step deviation improvement over the current policy. Without any closeness constraint, we hope this approach can allow larger policy improvement and hence lead to a faster convergence rate.

In summary, our main contribution in this thesis work is to design and analyze several reduction frameworks that reduces the hard RL problems to easier problems such as no-regret online learning and classic supervised learning, by introducing extra assumptions, including the availability of expert policies during training, the access to efficient model-based optimal control black-box solvers, and the ability to reset systems. Such additional assumptions and the reduction frameworks provide extra flexibility to Reinforcement Learning, which moves us one step closer to the generalization and efficiency of RL.
Chapter 2

Background

This thesis work focuses on sequential learning setting, where data samples arrive one by one and a learning algorithms has to learn and also predict on the fly. All of the algorithms we discuss in this work assume the data is generated from either of the following two settings: (1) a pure online learning setting where there is no statistical assumptions (e.g., Markovian) on the underlying data generation process (e.g., data could be generated in an adversarial way), (2) Markov Decision Processes (MDPs) where the data generated at this time step only depends on the data generated from the last time step, and such dependency is stochastic but fixed along the whole learning process. Below we first formally introduce the online learning setting in Sec. 2.1 and then discuss about the basics of MDPs in Sec. 2.2.

2.1 Online Learning

We focuses on Online Convex Programming (OCP) setting [Shalev-Shwartz et al., 2012, Zinkevich, 2003] where given a convex decision set $\mathcal{X}$, at the $n^{th}$ iteration, the learning algorithm needs to make a decision $x_n \in \mathcal{X}$ while simultaneously the nature prepares a convex loss function $\ell_n : \mathcal{X} \to \mathbb{R}$. Then both the learner and the nature reveals their choices: $x_n$ and $\ell_n$, and the learner suffers loss $\ell_n(x_n)$. Note that the process of nature preparing $\ell_n$ does not have to follow any statistical random process, and in fact could be generated adversarially. The goal of the learner is to be no-regret:

$$\sum_{n=1}^{N} \ell_n(x_n) - \min_{x^* \in \mathcal{X}} \sum_{n=1}^{N} \ell_n(x^*) = o(N), \quad (2.1)$$

where $o(N)$ stands for sublinear growth with respect to $N$ (e.g., $\sqrt{N} \in o(N)$). When one divides $N$ on both sides of Eq. 2.1 and take $N \to \infty$, we see that at:

$$\lim_{N \to \infty} \left[ \frac{1}{N} \sum_{n=1}^{N} \ell_n(x_n) - \min_{x^* \in \mathcal{X}} \sum_{n=1}^{N} \ell_n(x^*) \right] \leq 0, \quad (2.2)$$

namely in a long run, the learner is doing as well as the best decision it could have made if it knew all the loss functions in hindsight, thus the name of no-regret.

No-Regret Online Learning is an extremely popular and active research area. Classic no-regret online learning algorithms includes Online Gradient Descent (OGD) [Zinkevich].
A discounted, infinite-horizon Markov Decision Process (MDP) is defined as $(S, A, P, c, \rho_0, \gamma)$. Here, $S$ is a set of states, $A$ is a set of actions, and $P$ is the transition dynamics: $P(s'|s, a)$ is the probability of transitioning to state $s'$ from state $s$ by taking action $a$. We use $P_{s,a}$ in short for $P(s'|s, a)$. We denote $c(s, a)$ as the cost of taking action $a$ while in state $s$. Finally, $\rho_0$ is the initial distribution of states, and $\gamma \in (0, 1)$ is the discount factor.

We define a stochastic policy $\pi$ such that for any state $s \in S$, $\pi(\cdot|s) \in \Delta(A)$, where $\Delta(A)$ is the $A$-dimensional unit simplex. Conditioned on state $s$, $\pi(a|s) \in [0, 1]$ is the probability of taking action $a$ at state $s$. The distribution of states at time step $t$, induced by running the policy $\pi$ until and including $t$, is defined as $\forall s_t$:

$$d_t^\pi(s_t) = \sum_{\{s_i, a_i\}_{i \leq t-1}} \rho_0(s_0) \prod_{i=0}^{t-1} \pi(a_i|s_i) P(s_{i+1}|s_i, a_i),$$

where by definition $d_0^\pi(s) = \rho_0(s)$ for any $\pi$. Note that the summation above can be replaced by an integral if the state or action space is continuous. The state visitation distribution can be computed $d_\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t d_t^\pi(s)$. Denote $(d_\pi, \pi)$ as the joint state-action distribution such that $d_\pi(s, a) = d_\pi(s) \pi(a|s)$, then the expected total discounted sum of costs of a policy $\pi$ is:

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t, a_t) \sim d_\pi, \pi}[\gamma^t c(s, a)].$$

(2.3)

We define the value function $V^\pi(s)$ and state-action value function $Q^\pi(s, a)$ as

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{T} \gamma^t c(s_t, a_t) | s_0 = s, a \sim \pi \right],$$

$$Q^\pi(s, a) = c(s, a) + \gamma \mathbb{E}_{s' \sim P_{s,a}}[V^\pi(s')],$$

where the expectation in the first equation is taken with respect to the randomness from the policy $\pi$ and the MDP, and $P_{s,a}$ is short for $P(\cdot|s, a)$. With $V^\pi$ and $Q^\pi$, the advantage function $A^\pi(s, a)$ is defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. As we working in the cost setting, in the rest of the paper we refer to $A^\pi$ as the disadvantage function.

We consider episodic setting where at the $n^{th}$ iteration, the RL algorithm, denote as $B$, has to output a policy $\pi_n$. We measure the learning progress by regret:

$$R(B) = \sum_{n=1}^{N} J(\pi_n) - \sum_{n=1}^{N} J(\pi^*),$$

(2.4)

namely the cumulative performance difference from the optimal policy $\pi^*$. For discrete MDPs, we want the regret grows sublinearly in $N$ (e.g., $o(N)$), and we want polynomial dependency respect to the number of states $|S|$, number of actions $|A|$ and the effective horizon $1/(1 - \gamma)$. 

---

**References**

Note that we will also work on finite horizon setting, where we usually assume $\gamma = 1$, and we have fixed planning horizon $H$. Note that $H$ and $1/(1-\gamma)$ play the similar role in finite horizon setting and infinite horizon setting, respectively. For finite horizon setting, the value function and the state-action cost-to-go function will be indexed with time step $t$, i.e., $V^\pi_t(s_t) = E[\sum_{i=t}^{H} c(s_i, a_i) | a_i \sim \pi(\cdot|s)]$, and $Q^\pi_t(s_t, a_t) = c(s_t, a_t) + \gamma E_{s_{t+1} \sim P(.|s_t, a_t)} V^\pi_{t+1}(s_{t+1})$.

2.2.1 Partial Observable Markov Decision Process

In this thesis work, we will touch Partial Observable Markov Decision Process (POMDP) as well. Different from MDPs, POMDPs assumes that we cannot observe the state $s$. Instead, we have an observation function that generates a partial observation from an observation space $\mathcal{O}$ given a state: $o_t \sim O(\cdot|s_t)$. Define belief $b_t$ of a state given a sequence of history of actions and observations $\{a_1, a_2, ..., a_t\}$ as $b_t(s) \propto P(s|a_1, a_2, ..., a_t)$, i.e., the probability of ending up in state $s$ at time step $t$, given we executed actions $a_1, ..., a_{t+1}$ and observed observations $o_1, ..., o_t$. Different from MDPs, the optimal policy for POMDPs is defined in belief space: $\pi(a|b)$ is the probability of taking action $a$ given a belief $b$. In this thesis work, instead of directly learning to model and track beliefs [Hefny et al., 2015; Sun et al., 2016a,c; Venkatraman et al., 2016], as belief is determined by the history of actions and observations, we simply represent the policy $\pi(a|a_1, a_2, ..., a_{t-1}, o_t)$ by any Recurrent Neural Networks (RNN), such as Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]. We train RNN-based policies end-to-end in order to directly optimize the objective we care about (i.e., minimize the expected total cost). By training RNN-based policies in an end-to-end way, we hope the hidden states of RNNs can provide a compressed feature representation of the history that is tailed for optimizing the ultimate objective function. Previous work has shown that directly optimizing a recurrent structure based policy with respect to the objective function can work better than first training a model and then planning using the learned model [Hefny et al., 2018; Venkatraman et al., 2017].

2.2.2 Imitation Learning Setting

All the proposed IL approaches in this work assume that we are operating under some unknown MDP. In the IL setting, we further assume that we have access to some expert policy $\pi^e$. We define $Q^e(s, a)$ as the expert’s cost-to-go oracle, and $A^e(s, a)$ as the expert’s disadvantage oracle. We emphasize that $\pi^e$ may not be optimal, i.e., $\pi^e \notin \arg\min_{\pi} J(\pi)$. Throughout the paper, we assume $Q^e(s, a)$ is known or can be estimated without bias (e.g., by rolling out $\pi^e$: starting from state $s$, applying action $a$, and then following $\pi^e$ to the end). We emphasize here that $\pi^e$ is only available during training process and will not be available during test time, hence motivating us to learn from $\pi^e$ during training in order to mimic or even outperform $\pi^e$ during testing. Also, when we talking about IL, $\pi^e$ does not have to stand for a human expert (e.g., a human expert provides demonstrations by teleoperating a robot). In fact, $\pi^e$ could stand for near-optimal search algorithms via access to ground truth information [Chang et al., 2015b; Daume III et al., 2009] or motion planners or optimal control via access to a simulator during training [Choudhury et al., 2017a; Pan et al., 2017].

We consider two measures to quantify the performance of a learned policy $\pi$:

$$\Delta(\pi, \pi^e) = J(\pi) - J(\pi^e),$$

(2.5)
which measures the difference between the learned policy and the expert, and

\[ \Delta(\pi, \pi^*) = J(\pi) - J(\pi^*), \]  

(2.6)

which measures the performance gap between the learned policy and the optimal policy. Ideally, given a reasonably good expert policy \( \pi^e \) (\( \pi^e \) does not have to be optimal), we want to leverage \( \pi^e \) to quickly learn a policy that performs as good as \( \pi^e \), and then keep improving \( \pi \) to catch up \( \pi^* \).

Our main focus in this thesis is to improve the efficiency of RL. We proceed in three directions: we generalize previous Policy Evaluation algorithms [Baird, 1995, Sutton, 1988b] and provide new policy evaluation algorithms that can leverage arbitrary function approximators, guarantee to work without any statistical assumptions on the underlying sample generation process, and can take advantage of any faster No-Regret Online Learning (see definitions in Chapter 2) algorithms to speed up policy evaluation learning process.
Chapter 3

Online Bellman Residual and Temporal Difference Algorithms

3.1 Introduction

Reinforcement learning (RL) is an online paradigm for optimal sequential decision making where a agent interacts with environments, takes actions, receives reward and tries to maximize its long-term reward, a discounted sum of all the rewards that will be received from now on. An important part of RL is policy evaluation, the problem of evaluating the expected long-term rewards of a fixed policy. Temporal Difference (TD) learning [Sutton, 1988a] is perhaps the best known family of algorithms for policy evaluation. It has been observed that when combined with function approximation, TD may diverge and lead to poor prediction. The Residual Gradient (RG) was proposed [Baird, 1995] to address these concerns. RG attempts to minimize the Bellman Error (BE), typically with linear function approximation, using stochastic gradient descent. Comparison between the family of TD algorithms and RG has received tremendous attention, although most of the analyses heavily rely on certain stochastic assumptions of the environment such as that the sequence of observations are Markovian or from a static Markov Decision Process (MDP). For instance [Schoknecht and Merke, 2003] showed that TD converges provably faster than RG if the value functions are presented by tabular form. [Scherrer, 2010] shows that Bellman Residual minimization enjoys a guaranteed performance while TD does not in general when states are sampled from arbitrary distributions (off-policy) that may not correspond to trajectories taken by the system.

Schapire and Warmuth [1996] and Li [2008] provided worst-case analysis of long-term predictive error for variants of the linear TD and RG under a non-probabilistic online learning setting. Their results rely on the spectral analysis of a matrix that is related to specific update rules of the TD and RG algorithms under linear function approximation. Unfortunately, this approach makes it more difficult to extend their worst-case (assumption free) analysis to broader families of algorithms and representations that target Bellman and Temporal Difference errors.

Following [Schapire and Warmuth, 1996] and [Li, 2008]'s online learning framework, in this chapter, we present two simple, general connections between long-term predictive error and no-regret online learning that attempts to minimize BE and TD. The central idea is
that methods such as TD and RG should be fundamentally understood as online algorithms as opposed to standard gradient methods, and that one cannot simultaneously make consistent predictions in the sense of TD and BE while doing a poor job in terms of long-run predictions. Similar to Schapire and Warmuth [1996] and Li [2008], our analysis does not rely on any statistical assumptions about the underlying system. This allows us to analyze difficult scenarios such as MDP with transition probabilities changing over time or even with each transition chosen entirely adversarial.

The main contribution of the paper is the analysis of the connections between online long-term reward prediction and no-regret online learning. Particularly, the first analysis on BE shows that any no-regret and stable [Ross and Bagnell, 2011] online learning algorithms, when targeting optimizing BE, ensure small prediction error. The second analysis focuses on TD and shows that when applying the family of Online Mirror Descent (OMD) on TD, we can also achieve small prediction error. We additionally show that Implicit Online Learning is another proper algorithm that can be used for optimizing TD to achieve small prediction error. These two analysis consequently suggests a broad new family of algorithms. Particularly, our analysis on BE generalizes the RG algorithm from Baird [1995] in a sense that RG is a specific example of our family of algorithms that runs Online Gradient Descent (OGD) [Zinkevich, 2003] on a sequence of BE loss functions. For TD, our analysis generalizes TD∗(0) from Schapire and Warmuth [1996] by showing that running OGD—a special form of OMD, reveals the update rule of TD∗(0).

The contributions in this chapter were first presented in [Sun and Bagnell, 2016, 2015].

3.2 Preliminaries

We consider the sequential online learning model presented in [Li, 2008, Schapire and Warmuth, 1996] where no statistical assumptions about the sequence of observations are made. The sequence of the observations can either be Markovian as typically assumed in RL problem settings or even adversarial. We define the observation at time step $t$ as $x_t \in \mathbb{R}^n$, which usually represents the features of the environment at $t$. Throughout the paper, we assume that feature vector $x$ is bounded as $\|x\|_2 \leq X$. The corresponding reward at step $t$ is defined as $r_t \in \mathbb{R}$, where we assume that reward is always bounded $|r| \leq R \in \mathbb{R}^+$. Given a sequence of observations $\{x_t\}$ and a sequence of rewards $\{r_t\}$, the long-term reward at $t$ is defined as $y_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k$, where $\gamma \in [0, 1)$ is a discounted factor. Given a function space $F$ the learner chooses a predictor $f$ at each time step from $F$ for predicting long-term rewards. Throughout this paper, we assume that any prediction made by a predictor $f$ at a state $x$ is upper bounded as $|f(x)| \leq P \in \mathbb{R}^+$, for any $f \in F$ and $x$.

At time step $t = 0$, the learner receives $x_0$, initializes a predictor $f_0 \in F$ and makes prediction $y_0$ of $y_0$ as $f_0(x_0)$. Rounds of learning then proceeds as follows: the learner makes a prediction $\hat{y}_t$ of $y_t$ at step $t$ as $f_t(x_t)$; the learner then observes a reward $r_t$ and the next state $x_{t+1}$; the learner updates its predictor to $f_{t+1}$. This interaction repeats and is terminated after $T$ steps. Throughout this paper, we call this problem setting as online prediction of long-term reward.

We first define the signed Bellman Error at step $t$ for predictor $f_t$ as $b_t = f_t(x_t) - r_t - \gamma f_t(x_{t+1})$, which measures effectively how self consistent $f_t$ is in its predictions between time step $t$ and $t + 1$. We define the corresponding Bellman Loss at time step $t$ with respect to predictor $f$ as:

$$\ell_t^B(f) := (f(x_t) - r_t - \gamma f(x_{t+1}))^2. \quad (3.1)$$
We also define *signed Temporal Difference Error* (signed TD error) at step \( t \) for predictor \( f_t \) as \( d_t = f_t(x_t) - r_t - \gamma f_{t+1}(x_{t+1}) \). We define *TD\(^*\) Loss at step \( t \) as:

\[
\ell_d^t(f) := (f(x_t) - r_t - \gamma f_{t+1}(x_{t+1}))^2.
\] (3.2)

The *Signed Prediction Error* of long-term reward at \( t \) for \( f_t \) is defined as \( e_t = f_t(x_t) - y_t \) and \( e^*_t = f^*(x_t) - y_t \) for \( f^* \) accordingly. We will typically be interested in bounding the *Prediction Error* (PE) \( e_t^2 \) of a given algorithm in terms of the best possible PE. To lighten notation in the following sections, all sums over time indices implicitly run from 0 to \( T-1 \) unless explicitly noted otherwise.

### 3.3 Online Bellman Residual Algorithms

In this section, we establish a connection between optimizing Bellman loss and worst case long-term predictive error. Particularly, we show that optimizing Bellman loss with any *stable* and *no-regret* online algorithms ensures small prediction error for long-term reward prediction.

We first define the stability condition:

**Definition Online Stability:** For the generated sequence of predictors \( f_t \), we say the algorithm is online stable if:

\[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} (f_t(x_{t+1}) - f_{t+1}(x_{t+1}))^2 = 0.
\] (3.3)

Intuitively, the online stability means that on average the difference between successive predictors is eventually small. That is, the difference between \( f_t(x_{t+1}) \) and \( f_{t+1}(x_{t+1}) \) is small on average. Online stability is a general condition and does not severely limit the scope of the online learning algorithms. For instance, when \( f \) is linear, the definition of stability of online learning in [Saha et al., 2012] (see Eq. 3 in [Saha et al., 2012] and [Ross and Bagnell, 2011]) implies our form of online stability. In fact, we can show that many popular no-regret online learning algorithms including OGD, ONS, OWF, implicit online learning, and FTRL satisfy our online stability condition. We refer reader to [Sun and Bagnell, 2015] for the detailed study of the online stability condition for the above mentioned no-regret online algorithms.

Define \( e_t = f_t(x_{t+1}) - f_{t+1}(x_{t+1}) \), with the online stability condition, we now ready to state the main theorem:

**Theorem 3.3.1.** Assume a sequence of predictors \( \{f_t\} \) is generated by running some online algorithm on the sequence of Bellman loss \( \{\ell_t^b\} \). For any predictor \( f^* \in F \), the sum of prediction errors \( \sum e_t^2 \) can be upper bounded as:

\[
(1 - \gamma)^2 \sum e_t^2 \leq 2 \sum (b_t^2 - b_t^*2) + 2\gamma^2 \sum e_t^2 \\
+ 2(1 + \gamma)^2 \sum e_t^2 + M, 
\] (3.4)

where

\[
M = 2(\gamma + \gamma^2)(e_0^*2 - e_T^2) - (\gamma^2 - \gamma)(e_T^2 - e_0^2) .
\]
By running a no-regret and online stable algorithm, as $T \to \infty$, the average prediction error is then asymptotically upper bounded by a constant factor of the best possible prediction error in the function class:

$$
\lim_{T \to \infty} \frac{\sum e_t^2}{T} \leq \frac{2(1 + \gamma)^2 \sum e_t^2}{(1 - \gamma)^2 T}.
$$

(3.5)

The proof of the theorem only consists of easy application of telescoping tricks and Cauchy-Schwart inequalities. We refer readers to [Sun and Bagnell, 2015] for the detailed proof of the above theorem. We emphasize that the above analysis is independent of the particular form of function approximation.

When $e_t^* = 0, \forall t$, from Theorem 3.3.1, it is easy to see that no-regret rate of $(1/T) \sum (b_t^2 - b_{t+1}^2)$ and the online stability rate of $(1/T) \sum e_t^2$ together determine the rate of the convergence of $(1/T) \sum e_t^2$. When $T \to \infty$ and $\gamma \to 1$ (specifically when $\gamma \geq (1/\sqrt{2})$), our upper bound analysis in Eq. 3.5 is asymptotically tighter than the upper bound in [Li, 2008] (Eq. 12) provided for RG. Since a large number of popular no-regret online algorithms satisfy the online stability condition, our theorem essentially expends the family of algorithms that can be used to learn predictors of long-term rewards.

We emphasize here that stability of online algorithms is essential for our results—the no-regret property can be shown by counter-example to be insufficient to achieve low predictive error [Sun and Bagnell, 2015]:

**Theorem 3.3.2.** There exists a sequence of $\{f_t\}$ that is no-regret with respect to the loss functions $\{\ell_t(f)\}$, but no constant $C \in \mathbb{R}^+$ exists that makes the following inequality hold:

$$
\lim_{T \to \infty} \frac{\sum e_t^2}{T} \leq C \frac{\sum e_t^2}{T}.
$$

(3.6)

### 3.4 Online TD Learning via Online Mirror Descent and Implicit Online Learning

The analysis in Sec. 3.3 is general enough such that almost any existing no-regret online learning algorithm can be used for optimizing Bellman loss and ensures small prediction error on long-term rewards. Though we wish such a nice generalization also exists for TD, we could not establish it. Instead we show that a broad family of online learning algorithms—Online Mirror Descent (OMD), when applied to $\mathcal{TD}^*$ loss, ensures small prediction error similar in form to [Schapire and Warmuth, 1996]. We also show that implicit online gradient descent, a special form of implicit online learning, can also be used for optimizing $\mathcal{TD}^*$ loss. The proofs of the theorems presented in this section are in the appendix.

**Online Mirror Descent for $\mathcal{TD}^*$ loss**

Let us define $R(f)$ as a regularization and assume that $R(f)$ is both smooth and strongly convex function with respect to $f$ with norm $\| \cdot \|$, defined by the inner product associated with $\mathcal{F}$ as $\|f\|^2 = \langle f, f \rangle$. A function $R(f)$ is $\alpha$-smooth and $\beta$-strongly convex if and only if:

$$
\frac{\beta}{2} \| f_t - f_{t+1} \|^2 \leq R(f_t) - R(f_{t+1}) - \nabla R(f_{t+1})(f_t - f_{t+1}) \leq \frac{\alpha}{2} \| f_t - f_{t+1} \|^2.
$$

(3.7)

Without loss of generality, we assume that $R(f)$ is $1$-strongly convex (otherwise simply scale it) and $\alpha$-smooth function with respect to $f$ with norm $\| \cdot \|$. For instance, when $f$ is linear, $\|w\|^2/2$ is $1$-strongly convex and $1$-smooth. When applying OMD on TD* loss, we have the following update rule, which we denote as OMD-TD*:

$$f_t = \arg \min_f f_t + \frac{1}{\mu} R(f);$$

$$\theta_{t+1} = \theta_t + (\gamma \gamma_t - r_t - \gamma \gamma_{t+1}) \nabla f_t (\mathbf{x}_t).$$

Note that when we compute $f_t$ using Eq. 3.8, the RHS of Eq. 3.8 actually implicitly depends on $\hat{y}_t$, which is equal to $f_t(\mathbf{x}_t)$ and hence depends on $f_t$. Here, we assume that though $f_t$ appears on both sides of Eq. 3.8, we can still solve for $f_t$ from Eq. 3.8 as TD*(0) does. In practice, whether or not we can solve $f_t$ from Eq. 3.8 could depend on the form of $R(f)$. For instance, when $R(f) = \|f\|^2/2$ and $f$ belongs to a Reproducing Kernel Hilbert Space (RKHS) (e.g., linear function $f(x) = w^T x$), we can achieve closed-form update of $f_t$. In fact, when $f(x) = w^T x$, $R(w) = \|w\|^2$, it is easy to show the update rule from Eq. 3.8 reveals the TD*(0) algorithm.

The following theorem shows optimizing TD* loss with OMD ensures small long-term prediction error:

**Theorem 3.4.1.** With $\mu = O(\frac{1}{\sqrt{T}})$ and $\mathcal{F}$ being a RKHS, OMD-TD* (Eq. 3.8 and 3.9) has the following bound:

$$\sum e_t^2 \leq \frac{2 + 2\gamma^2}{1 - \gamma^2} \sum e_t^2 + O(\sqrt{T}).$$

For the average prediction error $\sum e_t^2 / T$, we have:

$$\lim_{T \to \infty} \frac{\sum e_t^2}{T} \leq \frac{2 + 2\gamma^2}{1 - \gamma^2} \sum e_t^2.$$  

### Implicit Online Learning for TD* Loss

The OMD framework generalizes quite a few popular online algorithms such as Online Gradient Descent, Normalized Exponential Gradient (normalized EG), OGD with lazy projection and $p$-norm algorithm [Shalev-Shwartz, 2011]. However, OMD is conceptually different from another family of online algorithms—Implicit Online Learning [Kulis et al., 2010]. Implicit online learning algorithms usually are more stable and robust compared to algorithms with explicit update rules. The idea of implicit update has been applied to classic TD [Tamar et al., 2014], where the authors show the algorithm with implicit update is more stable than classic TD in a sense that it is not sensitive to learning step size.

Briefly, given the sequence of loss $\ell_t(f)$, implicit online learning updates $f$ as $f_{t+1} = \arg \min_f \ell_t(f) + \frac{1}{\mu_t} D_R(f, f_t)$, where $D_R(f, f_t)$ is the Bregman divergence generated from regularization $R$. For special case where $f$ is in RKHS, TD* loss $\ell_t^*(f)$ is actually a quadratic loss with respect to $f$. Hence, we propose to apply the implicit Online Gradient Descent—one special form of implicit online learning, to TD* loss. Set $R(f) = \|f\|^2/2$, we have the following update rule:

$$f_{t+1} = \arg \min_f \ell_t^*(f) + \frac{1}{\mu_t} \|f_t - f\|_2.$$
Note that the above update rule is implicit since \( \hat{y}_{t+1} \) (buried in \( \ell_t^d \)) depends \( f_{t+1} \). Depending on the form of \( f \), we can achieve closed-form solution for \( f_{t+1} \) from Eq. (3.12).

Below, we demonstrate a closed-form update rule for linear function \( f(x) = w^T x \) with \( R(w) = \|w\|^2 \). Replace \( f \) with \( w \) in Eq. (3.12), take the derivative with respect to \( w \), set it to zero, and solve for \( w_{t+1} \), we will get:

\[
w_{t+1} = w_t - \mu_t \frac{w_t^T x_t - r_t - \gamma \hat{y}_{t+1}}{1 + \mu_t \|x_t\|^2} x_t.
\]

(3.13)

Note that \( \hat{y}_{t+1} \) implicitly depends on \( w_{t+1} \). To solve for \( w_{t+1} \), we first dot product \( x_{t+1} \) on both sides of the above equation (the LHS becomes \( \hat{y}_{t+1} \)), solve for \( \hat{y}_{t+1} \) and then substitute \( \hat{y}_{t+1} \) back to the equation and solve for \( w_{t+1} \). This gives us the following Implicit-TD* update step:

\[
w_{t+1} = w_t - \mu \frac{w_t^T x_t - r_t - \gamma w_{t+1}}{1 + \mu \|x_t - \gamma x_{t+1}\|^2} x_t,
\]

(3.14)

where \( b_t = (w_t^T x_t - r_t - \gamma w_{t+1}) \). The corresponding update rule for RKHS with kernel \( K(\cdot, \cdot) \) is:

\[
f_{t+1} = f_t - \mu \frac{f_t(x_t) - r_t - \gamma f_t(x_{t+1})}{1 + \mu K(x_t, x_t - \gamma x_{t+1})} b_t K(x_t, \cdot),
\]

(3.15)

where \( b_t = (f_t(x_t) - r_t - \gamma f_t(x_{t+1})) \).

Implicit-TD* has the following upper bound on PE:

**Theorem 3.4.2.** With \( \mu = O(\frac{1}{\sqrt{T}}) \) and \( \mathcal{F} \) being a RKHS, Implicit-TD* (Eq. (3.14) and (3.15) has the following bound:

\[
\sum e_{t}^2 \leq \frac{(1 + \gamma)^2(2 + 2\gamma^2)}{(1 - \gamma)^2} \sum e_{t}^2 + O(\sqrt{T}).
\]

(3.16)

For the average prediction error \( \sum e_{t}^2 / T \), we have:

\[
\lim_{T \to \infty} \frac{\sum e_{t}^2}{T} \leq \frac{(1 + \gamma)^2(2 + 2\gamma^2)}{(1 - \gamma)^2} \sum e_{t}^2 / T
\]

(3.17)

**3.4.1 Discussion**

The bound of OMD-TD* is the tightest compared to Implicit-TD* and RG. Though our OMD-TD* bound is not as tight as the one from [Schapire and Warmuth, 1996], our analysis is more general. Our bound of RG is asymptotically tighter than the one from [Li, 2008] when \( \gamma \to 1 \). Experimentally we find that Implicit-TD* performs really well, which indicates that our worst-case bound for Implicit-TD* may be not tight.

**3.5 Experiment**

We applied several online learning algorithms to two simulated policy evaluation problems: (1) Random Walk with a ring chain, which is a variant of the Hall problem introduced in [Baird, 1995], (2) PuddleWorld adopted from [Sutton and Barto, 1998a]. We tested several
Figure 3.1: Convergence of prediction error. We applied a set of online algorithms on Bellman loss $\{\ell_B(w)\}$ (dot line) and $\mathbb{T}\mathbb{D}^*$-loss functions $\{\ell^*_d(w)\}$ (solid line) for Random walk (left) and Puddle World (right).

popular no-regret and stable online learning algorithms, including implicit online gradient descent (implicit OGD), online Newton step (ONS) [Hazan et al., 2006], online Frank Wolf (OFW) [Hazan and Kale, 2012] and classic online gradient descent [Zinkevich, 2003], on both $\mathbb{T}\mathbb{D}^*$ loss and Bellman loss.

Fig. 3.1 shows the convergence of average prediction error with respect to number of time steps. We note that ONS and implicit OGD give good convergence speed in general. Throughout the experiments, we found that implicit OGD works well for both $\mathbb{T}\mathbb{D}^*$ loss and Bellman loss. Our experimental results also show that our approaches have the possibility to achieve smaller prediction error than $\mathbb{T}\mathbb{D}(0)$ (e.g., Fig. 3.1b). Note that when optimizing $\mathbb{T}\mathbb{D}^*$ loss, ONS and OFW actually achieve good performance, though our analysis on $\mathbb{T}\mathbb{D}^*$ loss currently does not support ONS or OFW.

The experiment results for RKHS can be found at [Sun and Bagnell, 2015], where we also demonstrated these algorithms on a simulated helicopter hover domain [Coates et al., 2008].
Chapter 4

Efficient Reinforcement Learning via Imitation

4.1 Introduction

Although conventional supervised learning of deep models has been pivotal in advancing performance in sequential prediction problems, researchers are beginning to utilize Reinforcement Learning (RL) methods to achieve even higher performance [Bahdanau et al., 2016; Li et al., 2016; Ranzato et al., 2015]. In sequential prediction tasks, future predictions often depend on the history of previous predictions; thus, a poor prediction early in the sequence can lead to high loss (cost) for future predictions. Viewing the predictor as a policy $\pi$, deep RL algorithms are able to reason about the future accumulated cost in sequential prediction problems. These approaches have dramatically advanced the state-of-the-art on a number of problems including high-dimensional robotics control tasks and video and board games [Schulman et al., 2015a; Silver et al., 2016].

In contrast with general reinforcement learning methods, imitation learning and related sequential prediction algorithms such as SEARN [Daumé III et al., 2009], DaD [Venkatraman et al., 2015], AggreVaTe [Ross and Bagnell, 2014], PSIM [Sun et al., 2016c], and LOLS [Chang et al., 2015b] reduce the sequential prediction problems to supervised learning by leveraging a (near) optimal cost-to-go oracle that can be queried for the next (near)-best prediction at any point during training. Specifically, these methods assume access to an oracle that provides an optimal or near-optimal action and the future accumulated loss $Q^*$, the so-called cost-to-go. For robotics control problems, this oracle may be a human expert guiding the robot during the training phase [Abbeel and Ng, 2004] or the policy from an optimal MDP solver [Choudhury et al., 2015b; Ross et al., 2011] that is either too slow to use at test time or leverages information unavailable at test time. For sequential prediction problems, an oracle can be constructed by optimization (e.g., beam search) or by a clairvoyant greedy algorithm [Chang et al., 2015a; Daumé III et al., 2009; Rhinehart et al., 2015; Ross et al., 2013] that, given the training data’s ground truth, is near-optimal on the task-specific performance metric (e.g., cumulative reward, IoU, Unlabeled Attachment Score, BLEU).

We stress that the oracle is only required to be available during training. Therefore, the goal of IL is to learn a policy $\hat{\pi}$ with the help of the oracle $(\pi^*, Q^*)$ during the training.

1Expert, demonstrator, and oracle are used interchangeably.
session, such that $\hat{\pi}$ achieves similar or better performance at test time when the oracle is unavailable. In contrast to IL, reinforcement learning methods often initialize with a random policy $\pi_0$ or cost-to-go estimate $Q_0$ that may be far from optimal. The optimal policy (or cost-to-go) must be found by exploring, often with random actions.

A classic family of IL methods is to collect data from running the demonstrator or oracle and train a regressor or classifier via supervised learning. These methods [Abbeel and Ng, 2004; Ratliff et al., 2006; Syed et al., 2008; Ziebart et al., 2008] learn either a policy $\pi^*$ or $Q^*$ from a fixed-size dataset pre-collected from the oracle. Unfortunately, these methods exhibit a pernicious problem: they require the training and test data to be sampled from the same distribution, despite the fact they explicitly change the sample policy during training. As a result, policies learned by these methods can fail spectacularly [Ross and Bagnell, 2010].

Interactive approaches to IL such as SEARN [Daumé III et al., 2009], DAgger [Ross et al., 2011], and AggreVaTe [Ross and Bagnell, 2014] interleave learning and testing to overcome the data mismatch issue and, as a result, work well in practical applications. Furthermore, these interactive approaches can provide strong theoretical guarantees between training time loss and test time performance through a reduction to no-regret online learning.

In this work, we introduce AggreVaTeD, a differentiable version of AggreVaTe (Aggregate Values to Imitate [Ross and Bagnell, 2014]) which allows us to train policies with efficient gradient update procedures. AggreVaTeD extends and scales interactive IL for use in sequential prediction and challenging continuous robot control tasks. We provide two gradient update procedures: a regular gradient update developed from Online Gradient Descent (OGD) [Zinkevich, 2003] and a natural gradient update [Bagnell and Schneider, 2003; Kakade, 2002], which is closely related to Weighted Majority (WM) [Littlestone and Warmuth, 1994], a popular no-regret algorithm that enjoys an almost dimension-free property [Bubeck et al., 2015].

AggreVaTeD leverages the oracle to learn rich policies that can be represented by complicated non-linear function approximators. Our experiments with deep neural networks on various robotics control simulators and on a dependency parsing sequential prediction task show that AggreVaTeD can achieve expert-level performance and even super-expert performance when the oracle is sub-optimal, a result rarely achieved by non-interactive IL approaches. The differentiable nature of AggreVaTeD additionally allows us to employ Recurrent Neural Network policies, e.g., Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], to handle partially observable settings (e.g., observe only partial robot state). Empirical results demonstrate that by leveraging an oracle, IL can learn much faster than RL.

In addition to providing a set of practical algorithms, we develop a comprehensive theoretical study of IL on discrete MDPs. We construct an MDP that demonstrates exponentially better sample efficiency for IL than any RL algorithm. For general discrete MDPs, we provide a regret upper bound for AggreVaTeD with WM, which shows IL can learn dramatically faster than RL. We provide a regret lower bound for any IL algorithm, which demonstrates that AggreVaTeD with WM is near-optimal.

To summarize the contributions of this chapter: (1) AggreVaTeD allows us to handle continuous action spaces and employ recurrent neural network policies for Partially Observable Markov Decision Processes (POMDPs); (2) understanding IL from a perspective that is related to policy gradient allows us to leverage advances from the well-studied RL policy gradient literature (e.g., gradient variance reduction techniques, efficient natural gra-

---

2 i.e., the regret bound depends on poly-log of the dimension of parameter space.
dient computation); (3) we provide a new sample complexity study of IL and compare to RL, showing that we can expect up to exponentially lower sample complexity. Our experimental and theoretical results support the proposition:

Imitation Learning is a more effective strategy than Reinforcement Learning for sequential prediction with near-optimal cost-to-go oracles.

The contributions in this chapter are first presented in [Sun et al., 2017, 2018].

4.2 Regret Analysis for IL with Perfect Oracle

How much faster can IL learn a good policy than RL? In this section we quantify the gap on discrete MDPs when IL can (1) query for an optimal $Q^*$ or (2) query for a noisy but unbiased estimate of $Q^*$. To measure the speed of learning, we look at the cumulative regret of the entire learning process, defined as $R_N = \sum_{n=1}^{N} (J(\pi_n) - J(\pi^*))$. A smaller regret rate indicates faster learning. Throughout this section, we assume the expert $\pi^*$ is optimal. We consider finite-horizon, episodic IL and RL algorithms.

4.2.1 Exponential Gap

We consider an MDP $\mathcal{M}$ shown in Fig. 4.1 which is a depth-K binary tree-structure with $S = 2^K - 1$ states and two actions $a_l, a_r$: go-left and go-right. The transition is deterministic and the initial state $s_0$ (root) is fixed. The cost for each non-leaf state is zero; the cost for each leaf is i.i.d sampled from a given distribution (possibly different distributions per leaf). Below we show that for $\mathcal{M}$, IL can be exponentially more sample efficient than RL.

**Theorem 4.2.1.** For $\mathcal{M}$, the regret $R_N$ of any finite-horizon, episodic RL algorithm is at least:

$$\mathbb{E}[R_N] \geq \Omega(\sqrt{SN}).$$

The expectation is with respect to random generation of cost and internal randomness of the algorithm. However, for the same MDP $\mathcal{M}$, with the access to $Q^*$, we show IL can learn exponentially faster:
Theorem 4.2.2. For the MDP $\mathcal{M}$, AggreVaTe [Ross and Bagnell, 2014] with FTL can achieve the following regret bound:

$$R_N \leq O(\ln(S)).$$  \hfill (4.2)

Fig. 4.1 illustrates the intuition behind the theorem. Assume during the first episode, the initial policy $\pi_1$ picks the rightmost trajectory (bold black) to explore. We query from the cost-to-go oracle $Q^*$ at $s_0$ for $a_l$ and $a_r$, and learn that $Q^*(s_0, a_l) < Q^*(s_0, a_r)$. This immediately tells us that the optimal policy will go left (black arrow) at $s_0$. Hence the algorithm does not have to explore the right sub-tree (dotted circle).

Next we consider a more difficult setting where one can only query for a noisy but unbiased estimate of $Q^*$ (e.g., by rolling out $\pi^*$ finite number of times). The above halving argument will not apply since deterministically eliminating nodes based on noisy estimates might permanently remove good trajectories. However, IL can still achieve a poly-log regret with respect to $S$, even in the noisy setting:

Theorem 4.2.3. With only access to unbiased estimate of $Q^*$, for the MDP $\mathcal{M}$, AggreVaTe [Ross and Bagnell, 2014] with WM can achieve the following regret with probability at least $1 - \delta$:

$$R_N \leq O\left(\ln(S)\left(\sqrt{\ln(S)} + \sqrt{\ln(2/\delta)}N\right)\right).$$  \hfill (4.3)

The detailed proofs of the above three theorems can be found in Appendix E,F,G in [Sun et al., 2017], respectively. In summary, for MDP $\mathcal{M}$, IL is exponentially faster than RL.

4.2.2 Polynomial Gap and Near-Optimality

We next quantify the gap in general discrete MDPs and also show that AggreVaTeD is near-optimal. We consider the harder case where we can only access an unbiased estimate of $Q^*_t$, for any $t$ and state-action pair. The policy $\pi$ is represented as a set of probability vectors $\pi^{s,t} \in \Delta(A)$, for all $s \in S$ and $t \in [H]$:

$$\pi = \{\pi^{s,t}\}_{s \in S, t \in [H]}.$$  

Theorem 4.2.4. With access to unbiased estimates of $Q^*_t$, AggreVaTeD with WM achieves the regret upper bound:

$$R_N \leq O\left(\frac{\ln(S)\sqrt{\ln(A)}N}{\sqrt{\ln(A)N}}\right).$$  \hfill (4.4)

Here $Q^e_{\text{max}}$ is the maximum cost-to-go of the expert. The total regret shown in Eq. 4.4 allows us to compare IL algorithms to RL algorithms. For example, the Upper Confidence Bound (UCB) based, near-optimal optimistic RL algorithms from [Jaksch et al., 2010], specifically designed for efficient exploration, admit regret $\tilde{O}(HS^\frac{1}{2} + HAN)$, leading to a gap of approximately $\sqrt{HS}$ compared to the regret bound of imitation learning shown in Eq. 4.4.

We also provide a lower bound on $R_N$ for the $H = 1$ case which shows the dependencies on $N, A, S$ are tight:

Theorem 4.2.5. There exists an MDP ($H=1$) such that, with only access to unbiased estimates of $Q^*$, any finite-horizon episodic imitation learning algorithm must have:

$$\mathbb{E}[R_N] \geq \Omega(\sqrt{\ln(A)N}).$$  \hfill (4.5)

$^3$Here we assume $Q^e_{\text{max}}$ is a constant compared to $H$. If $Q^e_{\text{max}} = \Theta(H)$, then the expert is no better than a random policy of which the cost-to-go is around $\Theta(H)$.
The proofs of the above two theorems regarding general MDPs can be found in Appendix H & I in [Sun et al., 2017]. In summary for discrete MDPs, one can expect at least a polynomial gap and a possible exponential gap between IL and RL.

4.3 Differentiable Imitation Learning

Policy based imitation learning aims to learn a policy \( \hat{\pi} \) that approaches the performance of the expert \( \pi^* \) at test time when \( \pi^* \) is no longer available. In order to learn rich policies such as LSTMs or deep networks [Schulman et al., 2015a], we derive a method related to policy gradients for imitation learning and sequential prediction. To do this, we leverage the reduction of IL and sequential prediction to online learning as shown in [Ross and Bagnell, 2014] to learn policies represented by expressive differentiable function approximators.

The fundamental idea in [Ross and Bagnell, 2014] is to use a no-regret online learner to update policies using the following loss function at each episode \( n \):

\[
\ell_n(\pi) = \frac{1}{H} \sum_{t=1}^{H} \mathbb{E}_{s_t \sim d_n} \left[ \mathbb{E}_{a \sim \pi(\cdot|s_t)} [Q^*_t(s_t, a)] \right].
\] (4.6)

The loss function intuitively encourages the learner to find a policy that minimize the expert’s cost-to-go under the state distribution resulting from the current learned policy \( \pi_n \). Specifically, [Ross and Bagnell, 2014] suggest an algorithm named AggreVaTe (Aggregate Values to Imitate) that uses Follow-the-Leader (FTL) [Shalev-Shwartz et al., 2012] to update policies:

\[
\pi_{n+1} = \arg \min_{\pi \in \Pi} \sum_{n=1}^{N} \ell_n(\pi),
\]

where \( \Pi \) is a pre-defined convex policy set. When \( \ell_n(\pi) \) is strongly convex with respect to \( \pi \) and \( \pi^* \in \Pi \), after \( N \) iterations AggreVaTe with FTL can find a policy \( \hat{\pi} \) with:

\[
\mu(\hat{\pi}) \leq \mu(\pi^*) - \epsilon_N + O(\ln(N)/N),
\] (4.7)

where \( \epsilon_N = [\sum_{n=1}^{N} \ell_n(\pi^*) - \min_{\pi \in \Pi} \sum_{n=1}^{N} \ell_n(\pi)]/N \). Note that \( \epsilon_N \geq 0 \) and the above inequality indicates that \( \hat{\pi} \) can outperform \( \pi^* \) when \( \pi^* \) is not (locally) optimal (i.e., \( \epsilon_n > 0 \)). Our experimental results support this observation.

A simple implementation of AggreVaTe that aggregates the values (as the name suggests) will require an exact solution to a batch optimization procedure in each episode. When \( \pi \) is represented by large, non-linear function approximators, the \( \arg \min \) procedure generally takes more and more computation time as \( n \) increases. Hence an efficient incremental update procedure is necessary for the method to scale.

To derive an incremental update procedure, we can take one of two routes. The first route, suggested already by [Ross and Bagnell, 2014], is to update our policy with an incremental no-regret algorithm such as weighted majority [Littlestone and Warmuth, 1994], instead of with a batch algorithm like FTRL. Unfortunately, for rich policy classes such as deep networks, no-regret learning algorithms may not be available (e.g., a deep network policy is non-convex with respect to its parameters). So instead we propose a novel second route: we directly differentiate Eq. 4.6, yielding an update related to policy gradient methods. We work out the details below, including a novel update rule for IL based on natural gradients.

Interestingly, the two routes described above yield almost identical algorithms if our policy class is simple enough: e.g., for a tabular policy, AggreVaTe with weighted majority yields the natural gradient version of AggreVaTeD described below. And, the two
routes yield complementary theoretical guarantees: the first route yields a regret bound for simple-enough policy classes, while the second route yields convergence to a local optimum for extremely flexible policy classes.

### 4.3.1 Online Gradient Descent

For discrete actions, the gradient of $\ell_n(\pi_\theta)$ (Eq. 4.6) with respect to the parameters $\theta$ of the policy is

$$\nabla_\theta \ell_n(\theta) = \frac{1}{H} \sum_{t=1}^{H} \mathbb{E}_{s_t \sim d_t^{\pi_\theta}} \sum_a \nabla_\theta \pi(a|s_t; \theta) Q_t^*(s_t, a). \quad (4.8)$$

For continuous action spaces, we cannot simply replace the summation by integration since in practice it is hard to evaluate $Q_t^*(s, a)$ for infinitely many $a$, so, instead, we use importance weighting to re-formulate $\ell_n$ (Eq. 4.6) as

$$\ell_n(\pi_\theta) = \frac{1}{H} \sum_{t=1}^{H} \mathbb{E}_{s_t \sim d_t^{\pi_\theta}, a_t \sim \pi(\cdot|s_t; \theta_n)} \frac{\pi(a_t|s_t; \theta)}{\pi(a_t|s_t; \theta_n)} Q_t^*(s_t, a_t) \quad (4.9)$$

With this reformulation, the gradient with respect to $\theta$ is

$$\nabla_\theta \ell_n(\theta) = \frac{1}{H} \mathbb{E}_{s_t \sim \rho_{\pi_\theta}} \sum_{t=1}^{H} \nabla_\theta \pi(a_t|s_t; \theta) Q_t^*(s_t, a_t)$$

$$= \frac{1}{H} \mathbb{E}_{s_t \sim \rho_{\pi_\theta}} \sum_{t=1}^{H} \nabla_\theta \ln(\pi(a_t|s_t; \theta_n)) Q_t^*(s_t, a_t). \quad (4.10)$$

The above gradient computation enables a very efficient update procedure with online gradient descent: $\theta_{n+1} = \theta_n - \eta_n \nabla_\theta \ell_n(\theta)|_{\theta = \theta_n}$, where $\eta_n$ is the learning rate.

### 4.3.2 Policy Updates with Natural Gradient Descent

We derive a natural gradient update procedure for imitation learning inspired by the success of natural gradient descent in RL [Bagnell and Schneider, 2003, Kakade, 2002, Schulman et al., 2015a]. Following [Bagnell and Schneider, 2003], we define the Fisher information matrix $I(\theta_n)$ using trajectory likelihood:

$$I(\theta_n) = \frac{1}{H^2} \mathbb{E}_{s_t \sim \rho_{\pi_\theta}} \nabla_{\theta_n} \log(\rho_{\pi_\theta}(\tau)) \nabla_{\theta_n} \log(\rho_{\pi_\theta}(\tau))^T, \quad (4.11)$$

where $\nabla_{\theta} \log(\rho_{\pi_\theta}(\tau))$ is the gradient of the log likelihood of the trajectory $\tau$ which can be computed as $\sum_{t=1}^{H} \nabla_{\theta} \log(\pi_\theta(a_t|s_t))$. Note that this representation is equivalent to the original Fisher information matrix proposed by [Kakade, 2002]. Now, we can use Fisher information matrix together with the IL gradient derived in the previous section (Eq. 4.10) to compute the natural gradient as $I(\theta_n)^{-1} \nabla_\theta \ell_n(\theta)|_{\theta = \theta_n}$, which yields a natural gradient update: $\theta_{n+1} = \theta_n - \mu_n I(\theta_n)^{-1} \nabla_\theta \ell_n(\theta)|_{\theta = \theta_n}$. 

21
Interesting, as we mentioned before, when the given MDP is discrete and the policy class is in a tabular representation, AggreVaTe with Weighted Majority [Littlestone and Warmuth, 1994] yields an extremely similar update procedure as AggreVaTeD with natural gradient. As Weighted Majority can speed up online learning (i.e., almost dimension free [Bubeck et al., 2015]) and AggreVaTe with Weighted Majority enjoys strong theoretical guarantees on the performance of the learned policy [Ross and Bagnell, 2014], this similarity provides an intuitive explanation why we can expect AggreVaTeD with natural gradient to speed up IL and learn a high quality policy.

4.4 Sample-based Practical Algorithms

In the previous section, we derived a regular gradient update procedure and a natural gradient update procedure for IL. Note that all of the computations of gradients and Fisher information matrices assumed it was possible to exactly compute expectations including $E_{a \sim d^*}$ and $E_{a \sim \pi(a|s)}$. In this section, we provide practical algorithms where we approximate the gradients and Fisher information matrices using finite samples collected during policy execution.

4.4.1 Gradient Estimation and Variance Reduction

We consider an episodic framework where given a policy $\pi_n$ at episode $n$, we roll out $\pi_n$ $K$ times to collect $K$ trajectories $\{\tau^n_i\}$, for $i \in [K]$, $\tau^n_i = \{s_t^{i,n}, a_t^{i,n}, ...\}$. For gradient $\nabla_\theta \ell_n(\theta)|_{\theta=\theta_n}$, we can compute an unbiased estimate using $\{\tau^n_i\}_{i \in [K]}$:

$$\hat{\nabla}_\theta = \frac{1}{HK} \sum_{i=1}^K \sum_{t=1}^H \theta_n \pi_n(a|s_t^{i,n})Q_t^*(s_t^{i,n}, a), \quad (4.12)$$

for discrete and continuous setting respectively.

When we can compute $V_t^*(s)$, we can replace $Q_t^*(s_t^{i,n}, a)$ by the state-action advantage function $A_t^*(s_t^{i,n}, a) = Q_t^*(s_t^{i,n}, a) - V_t^*(s_t^{i,n})$, which leads to the following unbiased and variance-reduced gradient estimation for continuous action setting [Greensmith et al., 2004]:

$$\hat{\nabla}_\theta = \frac{1}{HK} \sum_{i=1}^K \sum_{t=1}^H \theta_n \ln(\pi_n(a_t^{i,n}|s_t^{i,n}))A_t^*(s_t^{i,n}, a_t^{i,n}), \quad (4.14)$$

In fact, we can use any baselines to reduce the variance by replacing $Q_t^*(s_t, a_t)$ by $Q_t^*(s_t, a_t) - b(s_t)$, where $b(s_t): S \rightarrow \mathbb{R}$ is a action-independent function. Ideally $b(s_t)$ should be some function approximator that approximates $V_t^*(s_t)$. In our experiments, we test linear function approximator $b(s) = w^T s$, which is online learned using $\pi^*$’s roll-out data.

The Fisher information matrix (Eq. 4.11) is approximated as:

$$\hat{I}(\theta_n) = \frac{1}{HK} \sum_{i=1}^K \nabla \theta_n \log(\rho_{\pi_n}(\tau_i)) \nabla \theta_n \log(\rho_{\pi_n}(\tau_i))^T$$

22
Algorithm 1 AggreVaTeD (Differentiable AggreVaTe)

1: Input: The given MDP and expert $\pi^\ast$. Learning rate $\{\eta_n\}$. Schedule rate $\{\alpha_n\}$, $\alpha_n \rightarrow 0, n \rightarrow \infty$.
2: Initialize policy $\pi_{\theta_0}$ (either random or supervised learning).
3: for $n = 1$ to $N$ do
4:  Mixing policies: $\hat{\pi}_n = \alpha_n\pi^\ast + (1 - \alpha_n)\pi_{\theta_n}$.
5:  Starting from $\rho_{\theta_0}$, roll out by executing $\hat{\pi}_n$ on the given MDP to generate $K$ trajectories $\{\tau^n_i\}$.
6:  Using $Q^\ast$ and $\{\tau^n_i\}_i$, compute the descent direction $\delta_{\theta_n}$ (Eq. 4.12, Eq. 4.13, Eq. 4.14, or CG).
7:  Update: $\theta_{n+1} = \theta_n - \eta_n\delta_{\theta_n}$.
8: end for
9: Return: the best hypothesis $\hat{\pi} \in \{\pi_n\}_n$ on validation.

$$= S_n S_n^T, \quad (4.15)$$

where, for notation simplicity, we denote $S_n$ as a $d \times K$ matrix where the $i$’s th column is $\nabla_{\theta_n} \log(p_{\pi_{\theta_n}}(\tau_i))/C(\sqrt{K})$. Namely the Fisher information matrix is represented by a sum of $K$ rank-one matrices. For large policies represented by neural networks, $K \ll d$, and hence $I(\theta_n)$ a low rank matrix. One can find the descent direction $\delta_{\theta_n}$ by solving the linear system $S_n S_n^T \delta_{\theta_n} = \nabla_{\theta_n} \delta_{\theta_n}$ using Conjugate Gradient (CG) with a fixed number of iterations, which is equivalent to solving the above linear systems using Partial Least Squares [Phatak and de Hoog, 2002]. This approach is used in TRPO [Schulman et al., 2015a]. The difference is that our representation of the Fisher matrix is in the form of $S_n S_n^T$ and in CG we never need to explicitly compute or store $S_n S_n^T$ which requires $d^2$ space and time. Instead, we only compute and store $S_n$ (O(2d)) and the total computational time is still $O(K^2 d)$. The learning-rate for natural gradient descent can be chosen as $\eta_n = \sqrt{\delta_{KL}(\nabla_{\theta_n} \delta_{\theta_n})}$, such that $KL(\rho_{\pi_{\theta_{n+1}}}(\tau)||\rho_{\pi_{\theta_n}}(\tau)) \approx \delta_{KL} \in \mathbb{R}^+$.

4.4.2 Differentiable Imitation Learning: AggreVaTeD

Summarizing the above discussion, we present the differentiable imitation learning framework AggreVaTeD, in Alg. 1. At every iteration $n$, the roll out policy $\hat{\pi}_n$ is a mix of the expert policy $\pi^\ast$ and the current policy $\pi_{\theta_n}$, with mixing rate $\alpha$ ($\alpha_n \rightarrow 0, n \rightarrow \infty$): at every step, with probability $\alpha$, $\hat{\pi}_n$ picks $\pi^\ast$ and picks $\pi_{\theta_n}$ otherwise. This mixing strategy with the decay rate was first introduced in [Ross et al., 2011] for IL, and later on was used in sequence prediction [Bengio et al., 2015]. In Line 6, one can either choose Eq. 4.13 or the corresponding variance reduced estimation Eq. 4.14 to perform regular gradient descent, and choose CG to perform natural gradient descent. AggreVaTeD is extremely simple: we do not need to perform any data aggregation (i.e., we do not need to store all $\{\tau_i\}_i$ from all previous iteration); the computational complexity of each policy update scales in $O(d)$. When we use non-linear function approximators to represent the policies, the analysis of AggreVaTe from [Ross and Bagnell, 2014] will not hold, since the loss function $\ell_n(\hat{\theta})$ is not convex with respect to parameters $\hat{\theta}$. Nevertheless, as we will show in experiments, in practice AggreVaTeD is still able to learn a policy that is competitive with, and sometimes superior to, the oracle’s performance.
Figure 4.2: Performance (cumulative reward $R$ on y-axis) versus number of episodes ($n$ on x-axis) of AggreVaTeD (blue and green), experts (red), and RL algorithms (dotted) on different robotics simulators.

4.5 Experiment

We evaluate our algorithms on robotics simulations from OpenAI Gym [Brockman et al., 2016] and on Handwritten Algebra Dependency Parsing [Duyck and Gordon, 2015]. We report reward instead of cost, since OpenAI Gym by default uses reward and dependency parsing aims to maximize UAS score. As our approach only promises there exists a policy among all of the learned policies that can perform as well as the expert, we report the performance of the best policy so far: $\max\{\mu(\pi_1), ..., \mu(\pi_i)\}$. For regular gradient descent, we use ADAM [Kingma and Ba, 2014] which is a first-order no-regret algorithm, and for natural gradient, we use CG to compute the descent direction. For RL we use REINFORCE [Williams, 1992] and Truncated Natural Policy Gradient (TNPG) [Duan et al., 2016].

4.5.1 Robotics Simulations

We consider CartPole Balancing, Acrobot Swing-up, Hopper and Walker. For generating an expert, similar to previous work [Ho and Ermon, 2016], we used a Deep Q-Network (DQN) to generate $Q^*$ for CartPole and Acrobot (e.g., to simulate the settings where $Q^*$ is available), while using the publicly available TRPO implementation to generate $\pi^*$ for Hopper and Walker to simulate the settings where one has to estimate $Q^*$ by Monte-Carlo roll outs $\pi^*$.

Discrete Action Setting We use a one-layer (16 hidden units) neural network with ReLu activation functions to represent the policy $\pi$ for the Cartpole and Acrobot benchmarks. The value function $Q^*$ is obtained from the DQN [Mnih et al., 2015] and represented by a multi-layer fully connected neural network. The policy $\pi_{\theta_i}$ is initialized with common ReLu
Table 4.1: Performance (UAS) of different approaches on handwritten algebra dependency parsing. SL stands for supervised learning using expert’s samples: maximizing the likelihood of expert’s actions under the sequences generated by expert itself. SL-RL means RL with initialization using SL. Random stands for the initial performances of random policies (LSTMs and NN). The performance of DAgger with Kernel SVM is from [Duyck and Gordon 2015].

<table>
<thead>
<tr>
<th>Method</th>
<th>SL (LSTMs)</th>
<th>SL (NN)</th>
<th>RL (LSTMs)</th>
<th>RL (NN)</th>
<th>DAgger (LSTMs)</th>
<th>DAgger (NN)</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>0.924±0.10</td>
<td>0.851±0.10</td>
<td>0.826±0.09</td>
<td>0.386±0.1</td>
<td>0.257±0.07</td>
<td>0.227±0.06</td>
<td>0.832±0.02</td>
</tr>
<tr>
<td>Natural</td>
<td>0.935±0.10</td>
<td>0.890±0.10</td>
<td>0.924±0.10</td>
<td>0.345±0.1</td>
<td>0.257±0.07</td>
<td>0.241±0.07</td>
<td>0.853±0.01</td>
</tr>
</tbody>
</table>

For the scheduling rate \( \{\alpha_i\} \), we set all \( \alpha_i = 0 \): namely we did not roll-in using the expert’s actions during training. We set the number of roll outs \( K = 50 \) and horizon \( H = 500 \) for CartPole and \( H = 200 \) for Acrobot.

Continuous Action Setting  We test our approaches on two robotics simulators with continuous actions: (1) the 2-d Walker and (2) the Hopper from the MuJoCo physics simulator. Following the neural network settings described in [Schulman et al. 2015a], the expert policy \( \pi^* \) is obtained from TRPO with one hidden layer (64 hidden states), which is the same structure that we use to represent our policies \( \pi_\theta \). We set \( K = 50 \) and \( H = 100 \). We initialize \( \pi_\theta \) by collecting \( K \) expert demonstrations and then maximize the likelihood of these demonstrations (i.e., supervised learning). We use a linear baseline \( b(s) = w^T s \) for RL and IL.

Fig. 4.2c and 5.1f show the performance averaged over 5 random trials. Note that AggreVaTeD outperforms the expert in the Walker by 13.7\% while achieving 97\% of the expert’s performance in the Hopper problem. After 100 iterations, we see that by leveraging the help from experts, AggreVaTeD can achieve much faster improvement rate than the corresponding RL algorithms (though eventually we can expect RL to catch up). In Walker, we also tested AggreVaTeD without linear baseline, which still outperforms the expert but performed slightly worse than AggreVaTeD with baseline as expected.
4.5.2 Dependency Parsing For Handwritten Linear Algebra

We consider a sequential prediction problem: transition-based dependency parsing for handwritten algebra with raw image data Duyck and Gordon [2015]. The parsing task for algebra is similar to the classic dependency parsing for natural language Chang et al. [2015a] where the problem is modelled in the IL setting and the state-of-the-art is achieved by AggreVaTe with FTRL (using Data Aggregation). The additional challenge here is that the inputs are handwritten algebra symbols in raw images. We directly learn to predict parse trees from low level image features (Histogram of Gradient features (HoG)). During training, the expert is constructed using the ground-truth dependencies in training data. The full state \( s \) during parsing consists of three data structures: Stack, Buffer and Arcs, which store raw images of the algebraic symbols. Since the sizes of stack, buffer and arcs change during parsing, a common approach is to featurize the state \( s \) by taking the features of the latest three symbols from stack, buffer and arcs (e.g., Chang et al. [2015a]). Hence the problem falls into the partially observable setting, where the feature \( o \) is extracted from state \( s \) and only contains partial information about \( s \). The dataset consists of 400 sets of handwritten algebra equations. We use 80\% for training, 10\% for validation, and 10\% for testing. Note that different from robotics simulators where at every episode one can get fresh data from the simulators, the dataset is fixed and sample efficiency is critical.

The RNN policy follows the design from Sutskever et al. [2014]. It consists of two LSTMs. Given a sequence of algebra symbols \( \tau \), the first LSTM processes one symbol at a time and at the end outputs its hidden states and memory (i.e., a summary of \( \tau \)). The second LSTM initializes its own hidden states and memory using the outputs of the first LSTM. At every parsing step \( t \), the second LSTM takes the current partial observation \( o_t \) (\( o_t \) consists of features of the most recent item from stack, buffer and arcs) as input, and uses its internal hidden state and memory to compute the action distribution \( \pi(\cdot|o_1, \ldots, o_t, \tau) \) conditioned on history. We also tested reactive policies constructed as fully connected ReLu neural networks (NN) (one-layer with 1000 hidden states) that directly maps from observation \( o_t \) to action \( a \), where \( o_t \) uses the most three recent items. We use variance reduced gradient estimations, which give better performance in practice. The performance is summarised in Table 4.1. Due to the partial observability of the problem, AggreVaTeD with a LSTM policy achieves significantly better UAS scores compared to the NN reactive policy and DAgger with a Kernelized SVM Duyck and Gordon [2015]. Also AggreVaTeD with a LSTM policy achieves 97\% of optimal expert’s performance. Fig. 4.3 shows the improvement rate of regular gradient and natural gradient on both validation set and test set. Overall we observe
that both methods have similar performance. Natural gradient achieves a better UAS score in validation and converges slightly faster on the test set but also achieves a lower UAS score on test set.
Chapter 5

Generalizing Policy Iteration via Dual Policy Iteration

5.1 Introduction

Approximate Policy Iteration (API) \cite{Bagnell2004, BertsekasTsitsiklis1995, KakadeLangford2002, Lazareti2010, Scherrer2014}, including conservative API \cite{KakadeLangford2002}, API driven by learned critics \cite{RummeryNiranjan1994}, or gradient-based API with stochastic policies \cite{BagnellSchneider2003, BaxterBartlett2001, Kakade2002, Schulman2015}, have played a central role in Reinforcement Learning (RL) for decades. While the vanilla API has essentially no performance guarantees, methods that leverage more controlled updates \cite{Bagnell2004, KakadeLangford2002} can provide both local optimality guarantees and global guarantees that depend on how samples are generated (e.g., a uniform reset distribution). Successful API approaches can be thought of as making “small” changes: by conservatively mixing with previous policies \cite{KakadeLangford2002}, modifying only single time steps \cite{Bagnell2003, Kakade2002}, or making small changes to policy parameters \cite{Bagnell2003, Kakade2002}. Recently, a new class of API algorithms, which we call Dual Policy Iteration, has begun to emerge. These algorithms follow a richer pattern for improving the policy, with two policies under consideration at any time during training: a reactive policy, usually learned by some form of function approximation, used for generating samples and deployed at test time, and an intermediate policy that can only be constructed or accessed during training. For example, in Imitation Learning (IL), the second policy corresponds to an expert policy. \cite{RossBagnell2014, Sun2017} propose to update the reactive policy by performing policy iteration against the expert policy (i.e., use the state-action values of the expert policy) and show that it learns much faster than regular API. The intuition is that the expert policy, a policy that is better than the reactive policy, directly informs the reactive policy’s improvement direction thereby avoiding costly local random exploration. Although we do not assume access to an expert policy in RL, we can construct a similar intermediate “expert” policy at any time during training. For example, ExIt \cite{Anthony2017} maintains and updates a UCT-based policy (Upper Confidence Bound applied to Tree \cite{KocsisSzepesvari2006}) as an intermediate expert. ExIt then updates the reactive policy by imitating the tree-based policy, which we anticipate would be better than the reactive policy as
it involves a multi-step lookahead search. AlphaGo Zero [Silver et al., 2017] employs a similar strategy to achieve super-human performance at the ancient game of Go. Zucker and Maas [2009] leverage (brute-force) forward tree search to improve learning Tetris. While tree search (e.g., MCTS) is an excellent way to construct an intermediate “expert” policy with fully known dynamics, for applications with unknown dynamics, one can instead learn dynamics and then perform forward search using techniques such as Value Iteration and Differential Dynamic Programming (DDP) to construct the “expert” policy that improves upon the current reactive one. [Levine and Abbeel, 2014].

In this chapter, we provide a general framework for synthesizing and analyzing such Dual Policy strategies by considering a particular alternating optimization strategy. Specific strategies for each optimization direction offer a new family of approximate policy iteration methods. We construct a simple instance of our framework, where one policy is computed from Model-Based Optimal Control (MBOC) and the reactive policy with arbitrary function approximations is updated incrementally. The resulting actor-critic-like algorithm iteratively learns a local transition model and applies MBOC to compute a locally optimal policy around the reactive policy, together with a corresponding state-action value function (the critic), and then executes a policy update on the reactive policy (the actor).

To evaluate our approach, we demonstrate our algorithm on synthetic discrete MDPs, and also multiple classic continuous control tasks, including helicopter aerobatics [Abbeel et al., 2005] and multiple locomotion tasks from the MuJoCo physics simulator [Todorov et al., 2012], and show that our algorithm is extremely sample-efficient compared to classic API algorithms such as CPI, as well as more recent actor-critic baselines [Schulman et al., 2015b]. We also extend the framework to a robust policy optimization setting [Atkeson, 2012, Bagnell and Schneider, 2001] where one has access to multiple training environments, and the goal is to learn a single policy that can be deployed immediately on a test environment without further training.

Furthermore, we show a general convergence analysis that extends existing API theory to the DPI. In particular, we show monotonic improvement, where the improvement mainly consists of the independent improvement resulting from each optimization direction. When neither of the optimization procedures can improve, then we show that we have reached a local optimum. Our analysis is general enough to provide theoretical intuition for previous successful practical Dual Policy API algorithms such as Expert Iteration (ExIt) [Anthony et al., 2017]. Additionally, we provide a more concrete analysis in the setting where model-based OC is used. Our theorem considers how the model error would affect policy improvement, and indicates that we only need locally accurate dynamics, i.e., a model that accurately predicts next states under the current policy’s state-action distribution. Note that locally accurate dynamics are much easier to learn than global dynamics, as learning a global model suffers from a much greater degree of model bias (i.e., a single function approximator is not able to capture the true model over the entire space), and requires a dataset that covers the entire state space. In this sense, our analysis is similar in spirit to previous work that uses inaccurate models for policy optimization [Abbeel et al., 2006, Kolter and Ng, 2009].

5.2 Preliminaries

We consider Markov Decision Processes defined in Sec. 2.2. Throughout this chapter, we assume that we know the form of the cost function $c(s, a)$, but the transition dynamics $P$ is unknown. We emphasize that at least, from a theoretical perspective, finding the optimal
policy of the MDP with known cost function is as difficult as finding the optimal solution with unknown cost function, in terms of sample and computational complexity [Azar et al., 2017, Jaksch et al., 2010].

For two distributions $P_1$ and $P_2$, $D_{TV}(P_1, P_2)$ denotes the total variation distance, which is related to the $L_1$ norm as $D_{TV}(P_1, P_2) = \|P_1 - P_2\|_1/2$ (if we have a finite probability space) and $D_{KL}(P_1, P_2)$ denotes the KL divergence: $D_{KL}(P_1, P_2)$. Throughout the paper, we use the Performance Difference lemma (PDL) extensively:

**Lemma 5.2.1 (Lemma 6.1 [Kakade and Langford, 2002]).** For any two policies $\pi$ and $\pi'$, we have:

$$J(\pi) - J(\pi') = \frac{1}{1 - \gamma} E_{(s,a) \sim d_\pi} [A^\pi(s,a)] .$$ (5.1)

### 5.3 Dual Policy Iteration

In this section, we propose an alternating optimization framework for DPI, inspired by the PDL (Lemma 5.2.1). We first introduce the general alternating optimization framework and then discuss a simple instance that combines model-free and model-based updates.

Let us consider the following optimization framework:

$$\min_{\pi \in \Pi} \max_{\eta \in \Pi} E_{s \sim d_\pi} \left[ E_{a \sim \pi(\cdot|s)} [A^\eta(s,a)] \right] .$$ (5.2)

Note that the optimal solution for the above equation is $(\pi, \eta) = (\pi^*, \pi^*)$. To show this, we can simply apply the PDL, which reveals a min-min problem:

$$\arg \min_{\pi \in \Pi} \arg \max_{\eta \in \Pi} E_{s \sim d_\pi} \left[ E_{a \sim \pi(\cdot|s)} [A^\eta(s,a)] \right] = \arg \min_{\pi \in \Pi} J(\pi) - \arg \min_{\eta \in \Pi} J(\eta).$$ (5.3)

The above framework immediately proposes a general strategy for policy iteration: alternatively fix one policy and update the second policy. Mapping to previous practical Dual Policy API algorithms [Anthony et al., 2017, Silver et al., 2017], $\pi$ could stand for the fast reactive policy and $\eta$ could correspond to the tree search policy. Below we first consider how we can update $\pi$ given a fixed $\eta$. For notation purposes, we use $\pi_n$ and $\eta_n$ to represent the two policies in the $n$th iteration.

#### 5.3.1 Updating $\pi$

We update $\pi_n$ to $\pi_{n+1}$ by performing the following constrained optimization procedure:

$$\arg \min_{\pi} \mathbb{E}_{s \sim d_{\pi_n}} \left[ \mathbb{E}_{a \sim \pi(\cdot|s)} [A^{\eta_n}(s,a)] \right] ;$$ (5.4)

$$s.t., \mathbb{E}_{s \sim d_{\pi_n}} [D_{TV}(\pi(\cdot|s), \pi_n(\cdot|s))] \leq \beta$$ (5.5)

Note that our formulation in Eq. 5.4 is different from existing API algorithms (e.g., CPI) which use the disadvantage function $A^{\pi_n}(s,a)$ of the current policy $\pi_n$ while in our case, we use $\eta_n$’s disadvantage function. As Eq. 5.4 is performing API with respect to a fixed policy $\eta_n$, we can solve Eq. 5.4 by converting it to supervised learning problem such as cost-sensitive classification [Kakade and Langford, 2002], subject to an L1 constraint. It is
worth pointing out that the trust region definition in Eq. 5.5 is empirically measurable using samples from $d_{\pi_n}$ (i.e., executing $\pi_n$ on real systems to generate $s$), while the analysis in previous work (i.e., TRPO from [Schulman et al., 2015a]) used a much stronger definition of trust region in the format of $\max_{s \in S} D_{TV}(\pi(\cdot|s), \pi_n(\cdot|s))$, which is not empirically measurable unless one can enumerate all states.

It is interesting to note that a conservative policy update procedure can be viewed as approximately solving the above constrained optimization problem. Following CPI, we first solve Eq. 5.4 without considering the constraint (Eq. 5.5) to obtain a policy $\pi^*_n$. We then conservatively update $\pi_n$ to $\pi_{n+1}$ as:

$$\pi_{n+1} = (1 - \beta)\pi_n + \beta\pi^*_n.$$  

(5.6)

Note that $\pi_{n+1}$ satisfies the constraint as $D_{TV}(\pi_{n+1}(\cdot|s), \pi_n(\cdot|s)) \leq \beta, \forall s$. Intuitively, a conservative update can be understood as first solving the objective function to obtain $\pi^*_n$ without considering the constraint, and then move $\pi_n$ towards $\pi^*_n$ until the boundary of the region defined by the constraint is reached. For a parameterized policy, in Sec. 5.4.1 we will introduce a corresponding natural gradient update procedure.

### 5.3.2 Updating $\eta$

In this section, we present how to update $\eta$ given a fixed $\pi$. Given $\pi_n$, the objective function for $\eta$ in Eq. 5.2 becomes:

$$\max_{\eta} \mathbb{E}_{s \sim d_{\pi_n}} \left[ \mathbb{E}_{a \sim \pi_n(\cdot|s)} [A^\eta(s, a)] \right].$$  

(5.7)

While computing the functional gradient with respect to $\eta$ measured at $\eta_n$ is complicated, it can be estimated using, for example, finite differences. We propose a different approach that leverages the PDL. It is not difficult to see that updating $\pi^*$ is equivalent to finding the optimal policy $\pi^*$: $\arg \max_{\eta} (J(\pi_n) - J(\eta)) \equiv \arg \min_{\eta} J(\eta)$, regardless of what $\pi_n$ is. As directly minimizing the objective $J(\eta)$ is as hard as the original problem, we update $\eta$ locally by constraining it to a trust region around $\pi_n$:

$$\arg \min_{\eta} J(\eta),$$  

(5.8)

$$s.t., \mathbb{E}_{s \sim d_{\pi_n}} D_{TV}[(\eta(\cdot|s), \pi_n(\cdot|s))] \leq \alpha.$$  

(5.9)

### Updating via Model-based RL

The constrained optimization problem in Eq. 5.8 & 5.9 is nontrivial, as we do not know the transition dynamics $P_{s,a}$. If we did know the dynamics, then we could leverage any Model-Based Optimal Control (MBOC) algorithm to solve this problem. Therefore, we propose to learn $P_{s,a}$ from the samples generated by executing $\pi_n$ at the last optimization direction for $\pi$. Moreover, thanks to the trust region, we can simply learn a local dynamics model, under the state-action distribution $d_{s,a} \pi_n$.

We denote the optimal solution to the above constrained optimization (Eq. 5.8 & Eq. 5.9) under the real model $P_{s,a}$ as $\eta_{\pi_n}^*$. Note that, due to the definition of the optimality, $\eta_{\pi_n}^*$ must perform better than $\pi_n$: $J(\pi_n) - J(\eta_{\pi_n}^*) > \Delta_n(\alpha)$, where $\Delta_n(\alpha)$ is the performance gain from $\eta_{\pi_n}^*$ over $\pi_n$. Here the size of $\Delta_n(\alpha)$ depends on the size of the trust-region. When the

---

1In fact, [Schulman et al., 2015a] suggests relaxing $\max_{s \in S}$ to $\mathbb{E}_{s \sim d_{\pi_n}}$ for practical implementations of TRPO.
trust region expands, i.e., $\alpha$ increases, then $\Delta_n(\alpha)$ approaches the performance difference between the optimal policy $\pi^*$ and $\pi_n$.

To perform MBOC, we learn a locally accurate model—a model $\hat{P}$ that is close to $P$ under the state-action distribution induced by $\pi_n$: we seek a model $\hat{P}$, such that the quantity $\mathbb{E}_{(s,a) \sim d_{\pi_n}} D_{TV}(\hat{P}_{s,a}, P_{s,a})$ is small. Optimizing an $L_1$ objective may be not easy as it is not smooth, but note that, by Pinsker’s inequality, we have $D_{KL}(P_{s,a}, \hat{P}_{s,a}) \geq D_{TV}(\hat{P}_{s,a}, P_{s,a})^2$, which indicates that we can optimize a surrogate loss defined with KL-divergence:

$$\arg\min_{\hat{P} \in \mathcal{P}} \mathbb{E}_{s \sim d_{\pi_n}, a \sim \pi_n(s)} D_{KL}(P_{s,a}, \hat{P}_{s,a}) \leq D_{TV}(\hat{P}_{s,a}, P_{s,a})^2,$$

where we denote $\mathcal{P}$ as the model class. Hence we reduce the local model fitting problem into a classic maximum likelihood estimation (MLE) problem, where the training data $\{(s, a, s')\}$ can be easily collected by executing $\pi_n$ on the real system (i.e., $P_{s,a}$)!

For later analysis purposes, we denote $\hat{P}$ as the maximum likelihood estimator in Eq. 5.10 and assume $\hat{P}$ is $\delta$-optimal:

$$\mathbb{E}_{(s,a) \sim d_{\pi_n}} D_{TV}(\hat{P}_{s,a}, P_{s,a}) \leq \delta,$$

where $\delta \in \mathbb{R}^+$ is controlled by the complexity of model class $\mathcal{P}$ and by the amount of training data we sample using $\pi_n$, which can be analyzed by standard learning theory.

After achieving a locally accurate model $\hat{P}$, we can solve Eq. 5.8 using any existing stochastic MBOC solvers. Assume a MBOC solver returns an optimal policy $\eta_n$ under the estimated model $\hat{P}$ and the trust-region constraint:

$$\eta_n = \arg\min_{\pi} J(\pi),$$

s.t., \(s_{t+1} \sim \hat{P}_{s_{t},a_{t}}, \mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi, \pi_n) \leq \alpha \).

At this point, a natural question is: If $\eta_n$ is solved by an OC solver under $\hat{P}$, by how much can $\eta_n$ outperform $\pi_n$ when executed on real system $P$? Recall that the performance gap between the real optimal solution $\eta^*_n$ (the optimal under $P$) and $\pi_n$ is denoted as $\Delta_n(\alpha)$.

The following theorem quantifies the performance gap between $\eta_n$ and $\pi_n$:

**Theorem 5.3.1.** Assume $\hat{P}_{s,a}$ satisfies Eq. 5.11 and $\eta_n$ is the output of a MBOC solver for the optimization problem defined in Eq. 5.12 then we have:

$$J(\eta_n) \leq J(\pi_n) - \Delta_n(\alpha) + O \left( \frac{\gamma \delta}{1 - \gamma} + \frac{\gamma \alpha}{(1 - \gamma)^2} \right).$$

Theorem 5.3.1 indicates that when the model is locally accurate, i.e., $\delta$ is small (e.g., $\mathcal{P}$ is rich enough and we have enough data from $d_{\pi_n}$), $\alpha$ is small, and there exists a local optimal solution that is significantly better than the current policy $\pi_n$ (i.e., $\Delta_n(\alpha) \in \mathbb{R}^+$ is large), then the OC solver with the estimated model $\hat{P}$ finds nearly local-optimal solution $\eta_n$ that significantly outperforms $\pi_n$. 

5.3.3 Monotonic Improvement

In this section, we provide a general convergence analysis for the Dual Policy API framework we introduce above.

Let us define $A_n(\pi_{n+1})$ as the disadvantage of $\pi_{n+1}$ over $\eta_n$ under the state distribution $d_{\pi_n}$:

$$A_n(\pi_{n+1}) = \mathbb{E}_{s \sim d_{\pi_n}} \left[ \mathbb{E}_{a \sim \pi_{n+1}(\cdot | s)} [A_n^\eta(s, a)] \right],$$

Note that $A_n(\pi_{n+1})$ is at least as non-positive (if $\pi$ and $\eta$ are from the same function class, or $\pi$’s policy class is rich enough to include $\eta$), as if we set $\pi_{n+1}$ to $\eta_n$. In that case we simply have $A_n(\pi_{n+1}) = 0$, which means we can hope that the trust-region optimization procedure (Eq. 5.4) finds a policy $\pi_{n+1}$ that achieves $A_n(\pi_{n+1}) < 0$. The question we want to answer is: by how much is the performance of $\pi_{n+1}$ improved over $\pi_n$ by solving the two trust-region optimization procedures detailed in Eq. 5.8 and Eq. 5.4? Following Theorem 4.1 from Kakade and Langford [2002], we define $\varepsilon = \max_s |\mathbb{E}_{a \sim \pi_{n+1}(\cdot | s)} [A_n^\eta(s, a)]|$, which intuitively measures the maximum possible one-step improvement one can achieve from $\eta_n$. The following theorem states the performance improvement from $\pi_n$ to $\pi_{n+1}$:

**Theorem 5.3.2.** Solve Eq. 5.8 to compute $\eta_n$ and Eq. 5.4 to compute $\pi_{n+1}$. Then, the improvement of $\pi_{n+1}$ over $\pi_n$ is:

$$J(\pi_{n+1}) - J(\pi_n) \leq \frac{\beta \varepsilon}{(1 - \gamma)^2} - \frac{|A_n(\pi_{n+1})|}{1 - \gamma} - \Delta_n(\alpha).$$

(5.13)

When $\beta$ is small, we are guaranteed to find a policy $\pi_{n+1}$ where the total cost decreases by $\Delta_n(\alpha) + |A_n(\pi_{n+1})|/(1 - \gamma)$ compared to $\pi_n$. Note that classic API performance improvement [Kakade and Langford 2002; Schulman et al. 2015a] only contains a term that has the similar meaning and magnitude of the second term in the RHS of Eq. 5.13. Hence Dual Policy API boosts the performance improvement by introducing an extra term $\Delta_n(\alpha)$.

Theorem 5.3.2 simply assumes that $\eta_n$ is the optimal solution of Eq. 5.8 under real model $P_{s,a}$. When using MBOC with $\hat{P}$ to compute $\eta_n$ that approximates the true optimal solution, using Theorem 5.3.1 together with Theorem 5.3.2 we can easily show that that $\pi_{n+1}$ improves $\pi_n$ by:

$$J(\pi_{n+1}) - J(\pi_n) \leq -\Delta_n(\alpha) - \frac{|A_n(\pi_{n+1})|}{1 - \gamma} + O \left( \frac{\beta \varepsilon}{(1 - \gamma)^2} + \frac{\gamma \delta}{1 - \gamma} + \frac{\gamma \alpha}{(1 - \gamma)^2} \right).$$

When $\pi_n$ is far away from the optimal solution $\pi^*$, i.e., at the beginning of the learning process, one can expect $|\Delta_n(\alpha)|$ and $|A_n(\pi_{n+1})|$ to have large magnitude. When $|\Delta_n(\alpha)|$ is small, say $|\Delta_n(\alpha)| \leq \xi$ for some small positive real number $\xi$, then it means that $\pi_n$ is already an $\epsilon$-locally optimal solution, where we define a policy $\pi_n$ to be $\epsilon$-locally optimal if and only if there exists a positive real number $\alpha$ such that: $J(\pi_n) \leq J(\pi') + \epsilon, \forall \pi' \in \{ \pi : \mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi_n, \pi) \leq \alpha \}$. When $|A_n(\pi_{n+1})| \leq \xi$ also holds, then we can guarantee that $\eta_n$ and $\pi_n$ are good policies. Under the realizable assumption (i.e., II is rich):

$$\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi_n}} \left[ \mathbb{E}_{a \sim \pi_{n+1}(\cdot | s)} [A_n^\eta(s, a)] \right]$$
using the techniques from [Kakade and Langford, 2002], we can relate the performance of \( \eta_n \) to the optimal policy \( \pi^* \). We call a policy class \( \Pi \) closed under its convex hull if for any sequence of policies \( \{\pi_i\}_i, \pi_i \in \Pi \), the convex combination \( \sum_i w_i \pi_i \), for any \( w \) such that \( w_i \geq 0 \) and \( \sum_i w_i = 1 \), also belongs to \( \Pi \).

**Theorem 5.3.3.** Assume Eq. 5.14 holds and \( \Pi \) is closed under its convex hull, and

\[
\max \{ |\Delta_n(\pi_{n+1})|, \Delta(\alpha) \} \leq \xi \in \mathbb{R}^+,
\]

then for \( \eta_n \), we have:

\[
J(\eta_n) - J(\pi^*) \leq \left( \max_s \left( \frac{d^*(s)}{\rho_0(s)} \right) \right) \left( \frac{\xi}{\beta(1-\gamma)^2} + \frac{\xi}{\beta(1-\gamma)} \right).
\]

The term \( \max_s (d^*(s)/\rho_0(s)) \) measures the distribution mismatch between the initial state distribution \( \rho_0 \) and the optimal policy \( \pi^* \), and appears in previous API algorithms such as CPI [Kakade and Langford, 2002] and PSDP [Bagnell et al., 2004]. Although this shows that to guarantee good performance, one needs a \( \rho_0 \) similar to the best policy’s state visitation distribution or a uniform distribution, to the best of our knowledge this is the best performance guarantee in the API literature thus far. Combining the above discussion on \( \Delta_n(\alpha) \) and \( \Delta_n \) with Theorem 5.3.2, we show when either \( |\Delta_n(\alpha)| \) or \( |\Delta_n(\pi_{n+1})| \) have large magnitudes, \( \pi_{n+1} \) can improve over \( \pi_n \) significantly; when \( |\Delta_n(\alpha)| \) and \( |\Delta_n(\pi_{n+1})| \) are small, then \( \pi_n \) and \( \eta_n \) are already good policies.

### 5.3.4 Connection to Previous Work

The idea of computing a policy update guided by a better policy has been explored in practice under the setting where the dynamics is fully known and deterministic, where MCTS is leveraged to construct a policy that can perform better than the current reactive policy [Anthony et al., 2017, Silver et al., 2017]. For example, mapping to ExIt, our \( \eta_n \) plays the role of the tree-based policy, and our \( \pi_n \) plays the role of the apprentice policy. We force \( \eta \) to stay close to \( \pi \) with a trust region (for the purpose of the tractability of dynamics learning and MBOC), while ExIt ensures it by forming \( \eta \) as the weighted mixing of the search tree and the apprentice policy. One major difference in updating \( \pi \) is that ExIt uses DAGger [Ross et al., 2011] to update \( \pi \) by attempting to minimize the counts of mismatches with respect to \( \eta \), while we perform cost-sensitive classification with loss formed by the disadvantage vector \( A^\eta(s, \cdot) \in \mathbb{R}^{|A|} \), which enables us to link the imitation performance directly to the ultimate performance \( J(\pi) \), and allows \( \pi \) to achieve a one-step deviation improvement over \( \eta_n \) [Ross and Bagnell, 2014, Sun et al., 2017].

### 5.4 Unifying Model-free RL and Model-based RL: A Practical Algorithm

We have presented a unified alternating optimization framework and analysis for Dual Policy API. Here we develop a practical algorithm for the continuous state-action setting. We will focus on finite horizon problems with \( H \) denoted as the maximum possible horizon.
We denote the state space $S \subseteq \mathbb{R}^d$ and action space $A \subseteq \mathbb{R}^d$. We work on parameterized policies: we parameterize policy $\pi$ as $\pi(\cdot; \theta)$ for any $s \in S$ (e.g., a neural network with parameter $\theta$), and parameterize $\eta$ by a sequence of time-varying linear-Gaussian policies $\eta = \{\eta_t\}_{t \leq H}$, where $\eta_t(a|s) = \mathcal{N}(K_t s + k_t, P_t)$ with $K_t \in \mathbb{R}^{d_a \times d_a}$, $k_t \in \mathbb{R}^{d_a}$ and $P_t \in \mathbb{R}^{d_a \times d_a}$ as the parameters of $\eta$. We will use $\Theta = \{K_t, k_t, P_t\}_{0 \leq t \leq H}$ to represent the collection of the parameters of all the linear-Gaussian policies across the entire horizon. One approximation we make here is to replace the policy divergence measure as the parameters of all the linear-Gaussian policies across the entire horizon.

In our implementation, we use Conjugate Gradient with the Hessian-vector product trick [Schulman et al., 2015a] to directly compute $\nabla^2 D_{TV}(\pi, \pi_n)$ in Eq. 5.16. In practice, the empirical gradient $\theta_n$ could be nonlinear with respect to parameters $\theta$ of the policy $\pi$. Performing a second order Taylor expansion of the KL constraint around $\theta_n$, we get the following constrained optimization problem:

$$\min_{\theta} \mathbb{E}_{s \sim d_{\pi_n}} [\mathbb{E}_{a \sim \pi(\cdot; \theta)}[A^{\eta_\theta_n}(s, a)]]$$

subject to $$(\theta - \theta_n)^T F_{\theta_n}(\theta - \theta_n) \leq \beta,$$  \hspace{1cm} (5.16)

where $F_{\theta_n}$ is the Fisher information matrix or the Hessian of the trust-region constraint $\mathbb{E}_{s \sim d_{\pi_n}} D_{KL}(\pi_{\theta_n}, \pi)$, measured at $\theta_n$. Denote the objective $\mathbb{E}_{s \sim d_{\pi_n}} [\mathbb{E}_{a \sim \pi(\cdot; \theta)}[A^{\eta_\theta_n}(s, a)]]$ as $L_n(\theta)$, and denote $\nabla_\theta$ as $\nabla_{\theta} L_n(\theta)|_{\theta=\theta_n}$, then we can approximately optimize $\theta$ by performing a step of natural gradient descent (NGD) as

$$\theta_{n+1} = \theta_n - \mu F_{\theta_n}^{-1} \nabla_{\theta_n},$$  \hspace{1cm} (5.17)

where $\mu$ is set to $\sqrt{\beta/(\nabla_{\theta_n}^T F_{\theta_n}^{-1} \nabla_{\theta_n})}$ to ensure that the KL constraint is satisfied. The objective $L_n(\theta)$ could be nonlinear with respect $\theta$, depending on the function approximator used for $\pi_{\theta_n}$. Hence one step of gradient descent may not reduce $L_n(\theta)$ enough. In practice, we can perform $k$ steps ($k > 1$) of NGD shown in Eq. 5.17 with the learning rate shrinking to $\sqrt{\beta/k}/(\nabla_{\theta}^T F_{\theta_n}^{-1} \nabla_{\theta})$ to ensure that after $k$ steps, the solution still satisfies the constraint in Eq. 5.16. In our implementation, we use Conjugate Gradient with the Hessian-vector product trick [Schulman et al., 2015a] to directly compute $F_{\theta_n}^{-1} \nabla$.

Note that the unbiased empirical estimation of $\nabla_{\theta_n}$ and $F_{\theta_n}$ is well-studied in the literature and can be computed using samples generated from executing $\pi_{\theta_n}$. Assume we roll out $\pi_{\theta_n}$ to generate $K$ trajectories $\tau^i = \{s^i_0, a^i_0, ... s^i_T, a^i_T\}, \forall i \in [K]$. The empirical gradient and Fisher matrix can be easily formed using these samples as:

$$\nabla_{\theta_n} = \sum_{s,a} [\nabla_{\theta_n}(\ln(\pi(a|s; \theta_n))) A^{\eta_\theta_n}(s, a)]$$

$$F_{\theta_n} = \sum_{s,a} [(\nabla \ln(\pi(a|s; \theta_n)))(\nabla_{\theta_n} \ln(\pi(a|s; \theta_n)))^T].$$  \hspace{1cm} (5.18, 5.19)
Algorithm 2 AggreVaTeD-OC

1: **Input:** The given MDP
   Parameters $\alpha \in \mathbb{R}^+, \beta \in \mathbb{R}^+, k \geq 1, k \in \mathbb{N}$
2: Initialize $\pi_{\theta_0}$
3: for $n = 0$ to ...
4: \hspace{1em} Execute $\pi_{\theta_n}$ to generate a set of trajectories
5: \hspace{1em} Fit dynamics $\hat{P}$ using $\{s_t, a_t, s_{t+1}\}$ (Eq. 5.20)
6: \hspace{1em} Solve the minmax Lagrangian in Eq. 5.21 subject to learned dynamics $\hat{P}$ and obtain $\eta_{\theta_n}$
7: \hspace{1em} Form disadvantage $A_{\eta_{\theta_n}}$
8: \hspace{1em} Compute $\theta_{n+1}$ by $k$-steps of NGD (Eq. 5.17)
9: end for

5.4.2 Updating $\eta_{\theta_n}$

Now we introduce how to find $\eta_n$ given $\pi_n$ using model-based optimal control. In our implementation, we use Linear Quadratic Gaussian (LQG) optimal control [Kwakernaak and Sivan, 1972] as the black-box optimal control solver. We learn a sequence of time-dependent linear Gaussian transition models to represent $\hat{P}$:

$$s_{t+1} \sim \mathcal{N}(A_t s_t + B_t a_t + c_t, \Sigma_t), \forall t \in [1, T],$$  \hspace{1em} (5.20)

where $A_t \in \mathbb{R}^{d_s \times d_s}, B_t \in \mathbb{R}^{d_s \times d_a}, c_t \in \mathbb{R}^{d_s}$, $\Sigma_t \in \mathbb{R}^{d_s \times d_s}$ can be learned using classic linear regression techniques on a dataset $\{s_t, a_t, s_{t+1}\}$ collected from executing $\pi_n$ on the real system. Although the original stochastic dynamics $P(s, a)$ may be complicated over the entire space, a sequence of linear functions may be able to locally approximate it well (remember that our theorem only requires a locally accurate model $\hat{P}$ under distribution $d_{\pi_n}$).

Next, to find a locally optimal policy under the linear-Gaussian transitions (i.e., Eq. 5.12), we add the KL constraint to the objective with Lagrange multiplier $\mu$ and form an equivalent minmax problem:

$$\min_{\eta} \max_{\mu \geq 0} \mathbb{E} \left[ \sum_{t=1}^{T} \gamma^{t-1} c(s_t, a_t) + \mu \left( \sum_{t=1}^{T} \gamma^{t-1} \mathbb{E}_{s \sim d_{\pi_n}} [D_{KL}(\eta, \pi_n)] - \alpha \right) \right],$$  \hspace{1em} (5.21)

where $\mu$ is the Lagrange multiplier, which can be solved by alternatively updating $\pi$ and $\mu$ [Levine and Abbeel, 2014]. For a fixed $\mu$, using the derivation from [Levine and Abbeel, 2014], ignoring terms that do not depend on $\pi$, the above formulation can be written as:

$$\arg \min_{\eta} \mathbb{E} \left[ \sum_{t=1}^{T} \gamma^{t-1} (c(s_t, a_t)/\mu - \log \pi_n(a_t|s_t)) \right] - \sum_{t=1}^{T} \gamma^{t-1} \mathbb{E}_{s \sim d_{\pi_n}} [\mathcal{H}(\pi(\cdot|s))],$$  \hspace{1em} (5.22)

where $\mathcal{H}(\pi(\cdot|s)) = \sum_a \pi(a|s) \ln(\pi(a|s))$ is the negative entropy. Hence the above formulation can be understood as using a new cost function:

$$c'(s_t, a_t) = c(s_t, a_t)/\mu - \log(\pi_n(a_t|s_t)), \hspace{1em} (5.23)$$

Small $D_{KL}$ leads to small $D_{TV}$ by Pinsker’s inequality.
and an entropy regularization on $\pi$ that encourages the diversity of the actions induced by $\pi$. It is well known in the optimal control literature that when $c'$ is quadratic and dynamics are linear, the optimal sequence of linear Gaussian policies for the objective in Eq. 5.23 can be found exactly by a Dynamic Programming (DP) based approach, the Linear Quadratic Regulator (LQR) [Kwakernaak and Sivan, 1972]. Given a dataset $\{s_t, a_t\}$, $c'(s_t, a_t)$ collected from executing $\pi_n$, we can learn a quadratic approximation of $c'(s, a)$ [Levine and Abbeel, 2014]. With a quadratic approximation of $c'$ and linear dynamics, we solve Eq. 5.23 for $\eta$ exactly by LQR. Once we get $\eta$, we go back to Eq. 5.21 and update the Lagrange multiplier $\mu$, for example, by projected gradient ascent [Zinkevich, 2003]. Upon convergence, LQR gives us a sequence of controls in the format $\eta_{\theta_n} = \{N(K_t s_t + k_t, P_t); t \in [T]\}$, together with a sequence of quadratic cost-to-go functions $\{Q_t(s, a)\}_t$, where $Q_t$ is in the format of $[s, a]^T F_t [s, a]/2 + q_t^T[s, a] + v_t$. We use the cost-to-go to form the disadvantage function $A^{\pi_{\theta_n}}(s, a)$, which is in quadratic form as well.

If we treat $\eta$ as an intermediate expert, the update step is similar to AggreVaTeD—a differential IL approach [Sun et al., 2017]. Every iteration, we run $\pi_{\theta_n}$ on $P$ to gather samples of states and actions. We estimate locally linear dynamics $\hat{P}$ and then leverage an OC solver (e.g., LQR) to solve the Lagrangian in Eq. 5.21 to compute $\eta_{\theta_n}$ and $A^{\pi_{\theta_n}}$, subject to the learned dynamics $\hat{P}$. We then perform NGD. We summarize the procedure, AggreVaTeD-OC, in Alg. 2.

### 5.5 Application to Robust Policy Optimization

One application for our approach is robust policy optimization [Zhou et al., 1996]. We consider one particular robust policy optimization setting where we have multiple training environments that are all potentially different from, but similar to, the testing environments. The goal is to train a single policy using the training environments and deploy the policy on a test environment without any further training. Previous work suggests a policy that optimizes all the training models simultaneously is stable and robust during testing [Atkeson, 2012, Bagnell and Schneider, 2001], as the training environments together serve as a "regularization" to avoid overfitting to one particular training environment.

More formally, let us assume that we have $M$ training environments. At iteration $n$, we roll out $\pi_{\theta_n}$ on environment $i$ to generate a set of trajectories. For each environment $i$, following the MBOC approach introduced in Sec. 5.4.2 we learn a sequence of local linear Gaussian dynamics and compute a sequence of local linear Gaussian policies $\pi_{\theta_i}$ and their associated disadvantages $A^{\pi_{\theta_i}}$, $\forall i \in [M]$. With $A^{\pi_{\theta_i}}$, $\forall i \in [M]$, following the NGD update introduced in Sec. 5.4.1 we consider all training environments equally and formalize the objective $L_n(\theta)$ as $L_n(\theta) = \sum_{i=1}^{M} E_{a \sim d_{\pi_{\theta_i}}} [E_{a \sim \pi(s_t; \theta)} [A^{\pi_{\theta_i}}]]$. We update $\theta_n$ to $\theta_{n+1}$ by computing the gradient $\nabla_{\theta} L_n(\theta)|_{\theta=\theta_n}$ and perform $k$-step NGD. We output a single policy $\pi_{\theta_n}$ at the end that can potentially be used for different test environments.

### 5.6 Experiment

We tested our approach on several MDPs: (1) a set of random discrete MDPs (Garnet problems [Scherrer, 2014]) (2) Cartpole balancing [Sutton and Barto, 1998b], (3) Helicopter Aerobatics (Hover and Funnel) [Abbeel et al., 2005], (4) Swimmer, Hopper and Half-Cheetah

---

3Note $[s, a]$ stands for the vector concatenating $s$ and $a$, $F_t \in \mathbb{R}^{(d_s+d_a) \times (d_s+d_a)}$, $q_t \in \mathbb{R}^{d_s+d_a}$, $v_t \in \mathbb{R}$. 

37
from MuJoCo physics simulator \cite{Todorov2012}. The goals of these experiments are:
(a) to experimentally verify that using $\pi^\eta$ to perform API is more sample-efficient than using $\pi^\pi$. Although previous work, such as ExI, has compared against REINFORCE \cite{Williams1992} and experimentally provided an affirmative answer to this question, we would like to show the same phenomenon with $\eta$ computed from MBOC using learned local models;
(b) to show that our approach can be applied to robust policy search and can outperform existing approaches \cite{Atkeson2012}.

### 5.6.1 Comparison to CPI on Discrete MDP

Following \cite{Scherrer2014}, we randomly create ten discrete MDPs with 1000 states and 5 actions. Different from the techniques we introduced in Sec. 5.4.1 for continuous settings, here we use conservative update shown in Eq. [5.6] where each $\pi^\eta_n$ is a linear classifier and is trained using regression-based cost-sensitive classification \cite{Kakade2002}.

\footnote{Link to our implementation will be provide here.}
The feature for each state $\phi(s)$ is the binary encoding of the state ($\phi(s) \in \mathbb{R}^{\log_2(|S|)}$). We maintain the estimated transition $\hat{P}$ in a tabular representation. The policy $\eta$ is also in a tabular representation and is computed using exact Value Iteration under $\hat{P}$ and $c'(s, a)$ (hence we name our approach here as AggreVaTeD-VI). Note VI under $\hat{P}$ is slow when $|S|$ and $|A|$ are large, but we emphasize that this step does not require any extra samples, and efficient approximate VI (e.g., Gorodetsky et al. [2015]) techniques can be freely plugged in here. Fig. 5.1a reports the statistical performance of our approach and CPI over the 10 randomly created discrete MDPs. Note that our approach converges faster than CPI. The only difference between our implementation and CPI here is that we used $A^\eta$ instead of $A^\pi$ for the policy update. The results indicates that performing policy iteration against a better policy speeds up the learning process.

5.6.2 Comparison to Actor-Critic in Continuous Settings

We compare against TRPO-GAE on a set of continuous control tasks. TRPO-GAE is a actor-critic-like approach where both actor and critic are updated using trust region optimization. We use a two-layer neural network to represent policy $\pi$ which is updated by natural gradient descent. We use LQG as the underlying MBOC solver and we name our approach as AggreVaTeD-iLQG.

Fig. 5.1 (b-g) shows the comparison between our method and TRPO-GAE over a set of continuous control tasks (confidence interval is computed from 20 random trials). As we can see, our method is significantly more sample-efficient compared to TRPO-GAE. The major difference between our approach and TRPO-GAE is that we use $A^\eta$ while TRPO-GAE uses $A^\pi$ for the policy update. Note that both $A^\eta$ and $A^\pi$ are computed using the rollouts from $\pi$. The difference is that our approach uses rollouts to learn local dynamics and analytically estimates $A^\eta$ using MBOC, while TRPO-GAE directly estimates $A^\pi$ using rollouts. Overall, our approach converges faster than TRPO-GAE, which again indicates the benefit of using $A^\eta$ for API.

5.6.3 Experiments on Robust Policy Optimization

We consider two simulation tasks, cartpole balancing and helicopter funnel. For each task, we create ten environments by varying the physical parameters (e.g., mass of helicopter, mass and length of pole). We treat 7 of the environments for training and the remaining three for testing. We compare our algorithm against TRPO, which could be regarded as a model-free, natural gradient version of the first-order algorithm proposed in [Atkeson 2012]. We also ran our algorithm on a single randomly picked training environment but still tested the output on test environments, which is denoted as non-robust in Fig. 5.2. Fig. 5.2 summarizes the comparison between our approach and baselines. Similar to the trend we saw in the previous section, our approach is more sample-efficient in the robust policy optimization setup as well. It is interesting to see the “non-robust” approach failures to further converge, which shows the overfitting phenomenon: the learned policy overfits to one particular training environment, which hurts the testing performance.
Figure 5.2: Performance (mean in log-scale on y-axis) versus number of episodes ($n$ on x-axis) in robust control.
Chapter 6

Proposed Work

6.1 Proposed Work One: Temporal Difference Learning via Dynamic Programming

6.1.1 Limitations of Temporal Difference Learning

The classic Temporal Difference (TD) learning [Sutton 1988b] has several limitations:

- most of existing analysis of TD learning mainly focused on the asymptotic property (i.e., no finite sample analysis), and the analysis is limited to linear function approximations [Bertsekas and Tsitsiklis 1995, Tsitsiklis and Van Roy 1997];
- it has been shown that TD learning can fail to converge in off-policy setting [Tsitsiklis and Van Roy 1997];
- even in on-policy setting, TD learning with non-linear function approximation can fail to converge in worst case [Tsitsiklis and Van Roy 1997].

The online Bellman Residual algorithm we proposed in Chapter 3 fixed the first and the third limitation by providing finite sample analysis for arbitrary non-linear function approximation. However, one drawback of Bellman Residual-based algorithms is that in practice they often converge slower and converge to a worse solution than TD learning (if TD learning converges). Can we generalize classic TD learning so that it can work with arbitrary function approximation under both on-policy and off-policy setting, with finite sample analysis?

6.1.2 Problem Setup

A discounted and infinite-horizon MDP $\mathcal{M}$ and a policy $\pi$ together defines a Markov chain at state space $\mathcal{S}$, where $s_0 \sim \rho_0$, $s_{t+1} \sim P(\cdot|s_t, \pi(s_t))$ $\forall t \geq 0$. We consider a general setting where we have a behavior policy $\pi_b$ and a evaluation policy $\pi_e$, such that $\pi_b$ is used to generate data from which we want to compute the estimator for the value function of the evaluation policy $\pi_e$. When $\pi_b$ is equal to $\pi_e$, we call it on-policy policy evaluation, and off-policy policy evaluation otherwise. For any policy $\pi$, we will denote $d_{\pi,t}$ as the state.
f is a function approximator that aims to estimate the value-to-go of the evaluation policy $\pi$ distribution at time step $t$. We instead consider time-dependent state distribution $\pi$, generated from the behavior policy under the state distribution of $\pi$. We will maintain a sequence of function approximators $\{f_i\}_{i=0}^{H-1}$ to form a loss function $\ell_n(f_t) = (f_t(s) - r(s) + \gamma f^*_{t+1}(s'))^2$. The goal is to learn such $f(\cdot; \theta)$ only using trajectory $\{s, \pi_b(s), r, s'\}_{t=0}^{\infty}$ generated from the behavior policy $\pi_b$, such that $f(\cdot; \theta)$ is an accurate approximation of $V^{\pi_e}$ under the state distribution of $\pi_e$.

6.1.3 TD Learning via Dynamic Programming

In this section, we propose a new algorithm that leverages Dynamic Programming to perform temporal difference learning. We will maintain a sequence of function approximators $\{f_i(\cdot)\}_{i=0}^{H-1}$ where $f_i \in \mathcal{F}$, and will update each $f_i$ using the information from its next predictor $f_{i+1}$. As our algorithm will be iteratively updating each $f_i$, we denote $f_i^n$ as the function approximator at $t^{th}$ iteration. Algorithm 3 summarizes the TD-DP (Temporal Difference Learning via Dynamic Programming) algorithm. The algorithm proceeds in an iterative way. Every iteration, the algorithm receives a triple $(s, r(s), s')$, where we assume $s$ is generated from the stationary distribution of $\pi_b$, and $s'$ is generated by the evaluation policy: $s' \sim P(\cdot | s, \pi_e(s))$, and $r(s)$ is the reward at state $s$. At any time step $t$, with sample $\{s, r, s'\}$, we update the $t^{th}$ predictor $f_t$ using the next predictor $f_{t+1}^n$ by forming a loss function $\ell_n^i(f_t) = (f_t(s) - r - \gamma f^*_{t+1}(s'))$ and compute its gradient with respect to $f$ measured at $f_t$ as $\nabla f \ell_n^i(f_t)$ and let a first-order no-regret online optimizer $F_t$ updates $f_t^n$ to $f_t^{n+1}$. when at the last time step $H$, for the last predictor $f_H$, we simply use the reward $r_H$ to form $\ell_H^i(f) = (f(s) - r(s))^2$.
Analysis of On-Policy TD-DP

We first consider the on-policy case, i.e., the behavior policy and the evaluation policy are the same: \( \pi_b = \pi_e \). In this proposed work, we are going to focus on the simple setting where for each triple \((s, r(s), s')\), \(s \sim \mu\)—the stationary distribution of \( \pi_b \). Formally, \( \mu \) is the fixed point solution of the equation \( \mu(s) = \sum_{s' \in S} P(s|s', \pi_b(s')) \mu(s') \), \( \forall s \in S \).

Let us consider the last time step \( H \). The predictor \( f_H \) associated with the last time step \( H \) performs no-regret learning on the sequence of stochastic square loss \( \{(f_H(s_i) - r_i)^2\}_{i=1}^N \) where \( s_i \sim \mu \) and \( r_i = r(s_i, \pi_b(s_i)) \). Apply no-regret property, for \( \{f_H^n\}_{n=1}^N \), we have:

\[
\frac{1}{N} \sum_{n=1}^{N} (f_H^n(s_n) - r_n)^2 - \min_{f^* \in F} \frac{1}{N} \sum_{n=1}^{N} (f^*(s_n) - r_n)^2 \leq o(N)/N. \tag{6.1}
\]

Assume loss function \( (f(s) - r)^2 \) is strongly convex with respect to the predictor \( f \), adding expectation on both sides, we have:

\[
\mathbb{E}_{s \sim \mu}[(f_H^N(s) - r(s, \pi_b(s)))^2] \leq \min_{f^* \in F} \mathbb{E}_{s \sim \mu}[(f^*(s) - r(s, \pi_b(s)))^2] + o(N)/N. \tag{6.2}
\]

where the LHS of the above inequality quantifies the performance guarantee of \( f_H^N \), the first term of RHS of the above inequality quantifies the minimum possible prediction error we can get if we pick the best predictor from our hypothesis class; the second term of the RHS is the no-regret rate, and typically decreases at the rate of \( O(\log(N)/N) \) if assuming strong convexity. For notation simplicity, we denote the minimum possible prediction error one can achieve from the best possible hypothesis in \( F \) as

\[
\epsilon_H = \sqrt{\min_{f \in F} \mathbb{E}_{s \sim \mu}[(f(s) - r(s, \pi_b(s)))^2]}, \tag{6.3}
\]

and no-regret rate for \( f_H \) as \( \gamma_N = o(N)/N \) (note \( \gamma_N \to 0 \) as \( N \to \infty \)). Hence the regression risk at time step \( H \) is

\[
\hat{\delta}_H = \mathbb{E}_{s \sim \mu}[f_H^N(s) - r(s, \pi_b(s))]^2 \leq \epsilon_H^2 + \sqrt{\gamma_N}. \tag{6.4}
\]

Now we can analyze the performance of the second last predictor \( f_{H-1} \). We are interested in bounding the prediction error \( \epsilon_{H-1} = \sqrt{\mathbb{E}_{s \sim \mu}[(f(s) - V(s))^2]} \). Using the definition of \( V(s) = r(s) + \gamma \mathbb{E}_{s' \sim P(s, \pi_b(s))} V(s') \), we have:

\[
\epsilon_{H-1} \leq \mathbb{E}_{s \sim \mu}[(f_{H-1}^N(s) - r(s) - \gamma \mathbb{E}_{s' \sim P(s, \pi_b(s))}[f_H^N(s')])^2] + \gamma \mathbb{E}_{s \sim \mu}[(f_H^N(s) - V(s))^2] + \hat{\delta}_{H-1} + \gamma \epsilon_H. \tag{6.5}
\]

which essentially provides a recursion, and allow us to show that for \( f_0 \), we have:

\[
\epsilon_0 \leq \frac{1}{1 - \gamma} \max_{i=0}^{H-2} \hat{\delta}_i^2 + \gamma^{H-1} \epsilon_H. \tag{6.6}
\]

\[\text{If the loss function is just convex with respect to } f, \text{ then we can only guarantee good performance for the average } \bar{f} = (f_H^1 + \ldots + f_H^n)/N \text{ in theory. But in practice, one simply outputs and uses the last predictor } f_H.\]
where the $\max\{\delta_i\}$ serves as the maximum regression risk one had during training the predictors, which plays a similar role as the Bellman Residual. Given any pre-defined prediction threshold $\xi$, we can set $H = O\left(\frac{\log(1/\xi)}{1-\gamma}\right)$, to make sure that the predictor $f_0$ can be accurate up to:

$$O\left(\xi + \frac{1}{1-\gamma} \max\{\hat{\delta}_i\}_{i=0}^{H-1}\right). \tag{6.7}$$

Though we are able to upper bound the prediction error using the regression risks $\hat{\delta}_i$ which plays a similar role as Bellman error and could be small when the hypothesis class is rich and we have enough samples. However, ideally, we want to upper bound the predictor error by the smallest prediction error $\epsilon^*$ one can achieve from the hypothesis class $\mathcal{F}$. We leave the analysis as future work.

### Analysis of Off-Policy TD-DP

We analyze the performance of TD-DP in the off-policy setting, i.e., $\pi_b \neq \pi_e$. The analysis of off-policy is more challenging as when $\pi_b \neq \pi_e$, for a $(s, r(s), s')$, though $s$ is still i.i.d sampled from the stationary distribution $\mu$ of $\pi_b$, as $s'$ is sampled by executing $\pi_e(s)$, the distribution where $s'$ is sampled in general is neither the stationary distribution $\mu_b$ of $\pi_b$ nor the stationary distribution of $\pi_e$. In fact $s'$ is sampled from $P_{\mu_b}$ where $P$ is the transition matrix where $P_{j,i} = P(j|i, \pi_e(i))$. Without any assumptions, it is hard to prove anything regarding the prediction ability of off-policy algorithms (e.g., when $P(j|i, \pi_e(i))$ can visit a set of states that $\mu_b$ has zero probabilities over. ). One assumption that we likely will need is that $\mu_b$ has the probability support over the entire state space: $\mu_b(s) \neq 0, \forall s$.

We claim that different from classic off-policy TD algorithm that will diverge in the worst case, our finite horizon TD-DP algorithm will never diverge. This comes from the fact that our TD-DP terminates DP with fixed number of iterations, which results a finite sequence of regression risks $\{\hat{\delta}_i\}_{i=1}^H$, and we known that for a finite sequence of real numbers, the maximum among the sequence exists. Such a maximum can at least be used to upper bound the prediction error. On the other hand, if we operate DP with infinite many iterations, we will end up having an infinite long sequence of Bellman errors, whose maximum could be undefined (e.g., consider a sequence of real numbers $\{i\}_{i=1}^\infty$).

We leave the detailed analysis on off-policy setting and implementations of our TD-DP as future work.

### 6.2 Proposed Work Two: Advantage Weighted Model-Free Imitation Learning

#### 6.2.1 Problem Definition

We denote expert as $\pi^*$ and for simplicity throughout this section we assume that $\pi^*$ is the optimal policy and our policy class $\Pi$ is rich enough to include $\pi^*$: $\pi^* \in \Pi$. We assume that during training, we can query the expert $\pi^*$ to obtain. We assume that we have access to the state-action visitation from the expert policy $\pi^*$, denoted as $d_{\pi^*}$. Note that in practice such a distribution can be empirically approximated by a set of state-action pairs $\{s_n, a_n\}_{n}$, where $s_n$ are states generated by executing $\pi^*$ on the real systems, and $a_n$ is sampled according to the expert policy $\pi^*$ at state $s_n$. Note that such a dataset can be pre-collected.
6.2.2 Non-Interactive Imitation via Advantage Weighted Classification

estimation

One can easily verify the above equality holds as when

during training. We consider model-free setting where the transition dynamic

beforehand, while the expert policy \( \pi^* \) itself may not be available for further query anymore
during training. We consider model-free setting where the transition dynamic \( P(s'|s,a) \) is
unknown (i.e., we can only sample \( s' \) from \( P(.|s,a) \)).

Recall that the Performance Difference Lemma indicates that for any policy \( \pi \), we have:

\[
J(\pi^*) - J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi^*}} [A^\pi(s, \pi^*(s))].
\] (6.8)

For the purpose of analysis simplicity, we are going to directly work with the expert’s
state-action visitation \( d_{\pi^*} \). But we emphasize that whenever the notation \( \mathbb{E}_{(s,a) \sim d_{\pi^*}} f(s,a) \)
appears for any function \( f: S \times A \rightarrow \mathbb{R} \), we can always replace it by the unbiased empirical
estimation \( \sum_{(s,a) \in D^*} f(s,a) \).

6.2.2 Non-Interactive Imitation via Advantage Weighted Classification

Starting from Eq. (6.8) we can rewrite the right hand side by introducing a zero-one loss:

\[
J(\pi^*) - J(\pi) = \frac{1}{1-\gamma} E_{s \sim d_{\pi^*}} [A^\pi(s, \pi^*(s))1[\pi(s) \neq \pi^*(s)]].
\] (6.9)

One can easily verify the above equality holds as when \( \pi(s) = \pi^*(s) \) for a state \( s \), we simply
have \( A^\pi(s, \pi^*(s)) = 0 \). Note that the ultimate goal is to final a policy that maximizes the
objective \( J(\pi) \), hence we have:

\[
\arg \max_{\pi \in \Pi} J(\pi) = \arg \min_{\pi \in \Pi} [J(\pi^*) - J(\pi)] \\
= \arg \min_{\pi \in \Pi} E_{s \sim d_{\pi^*}} [A^\pi(s, \pi^*(s))1[\pi(s) \neq \pi^*(s)]] = \arg \min_{\pi \in \Pi} g(\pi),
\] (6.10)

where we denote \( g(\pi) = \mathbb{E}_{s \sim d_{\pi^*}} [A^\pi(s, \pi^*(s))1[\pi(s) \neq \pi^*(s)]] \).

At this point, it seems that we have reduced the RL problem to a simple weighted classification
problem where states are sampled from \( d_{\pi^*} \), and classification error is weighted
by \( A^\pi(s, \pi^*(s)) \)! However it is much harder than a weighted classification problem as the
weight \( A^\pi(s, \pi^*(s)) \) depends on the policy \( \pi \) which is the optimization parameter of the
functional \( g(\pi) \) ! Though optimizing \( g(\pi) \) is equivalent to directly optimizing \( J(\pi) \), we cannot
directly apply any off-shelf weighted classification algorithms to optimize \( g \).

An intuitive strategy is to leverage an iterative algorithm, where \( n^{th} \) iteration, we fix the
weight using the latest policy \( \pi_n \) as \( A^{\pi_n}(s, \pi^*(s)) \) for any \( s \sim d_{\pi^*} \). Let us define \( \ell_n(\pi) \) as:

\[
\ell_n(\pi) = E_{s \sim d_{\pi^*}} [A^{\pi_n}(s, \pi^*(s))1[\pi(s) \neq \pi^*(s)].
\] (6.11)

We then compute \( \pi_{n+1} \) by optimizing \( \ell_n(\pi) \), and then we move to the next iteration. For
stability purpose, we may need to be careful when computing \( \pi_{n+1} \) by making sure that
\( \pi_{n+1} \) is not that “far” away from \( \pi_n \) to make sure \( A^{\pi_n} \) and \( A^{\pi_{n+1}} \) is not that different. Below
we formalize this stability related intuition by a reduction to No-Regret Online Learning
(see Chapter 2 for the introduction of No-Regret Online Learning).
We then can i.i.d sample any number of examples from \( \tilde{\text{Leader}} \) Shalev-Shwartz et al. [2012] as the underlying no-regret online learner. The FTL \( \{ \pi_l \} \), where 

\[
\nu = \sum_i d_{\pi^*}(s_i) \max \left( 0, A^{\pi^*}(s_i, \pi^*(s_i)) \right),
\]

where \( \nu_n = \sum_i d_{\pi^*}(s_i) \max \left( 0, A^{\pi^*}(s_i, \pi^*(s_i)) \right) \) is the normalization constant to make sure \( d_{\pi^*} \) is a valid distribution over \( S \). Given a set of finite samples \( \{ s_i \}_{i=1}^M \) from \( d_{\pi^*} \), with weight for each \( s_i \) as \( w_i = A^{\pi^*}(s_i, \pi^*(s_i)) \), we can use well-established bootstrap technique to generate samples from \( d_{\pi^*} \). We can then approximate \( d_{\pi^*} \) by a empirical distribution \( \hat{d}_M(A) = \frac{1}{\sum_i w_i} \sum_{i=1}^M 1[s_i \in A] \) for any \( A \subseteq S \), i.e., we put mass \( w_i / \sum_i w_i \) at each \( s_i \).

We then can i.i.d sample any number of examples from \( \hat{d}_M \). Note that \( \hat{d}_M \) is a consistent estimator of \( d_{\pi^*} \).

Given \( \beta \in (0, 1) \), let us define a mixed state distribution \( \hat{d}_{\pi^*} \) as:

\[
\hat{d}_{\pi^*}(s) = \beta d_{\pi^*}(s) + (1 - \beta)d_{\pi^*} \cdot \nu_n,
\]

which is a mixed from the expert’s state distribution \( d_{\pi^*} \) and weighted expert’s state distribution \( d_{\pi^*} \). Using \( \hat{d}_{\pi^*} \), let us re-define the loss function \( \ell_n(\pi) \) as:

\[
\ell_n(\pi) = \mathbb{E}_{s \sim \hat{d}_{\pi^*}}[l(\pi(s), \pi^*(s))],
\]

where \( l(a, a') \) is a convex surrogate loss function that upper bounds the zero-one loss \( 1[\pi^*(s) \neq \pi(s)] \) (e.g., \( l(a, a') \) could be hinge loss).

The reduction to no-regret online learning works as follows. With some policy \( \pi_0 \) as the initialization, and with access to any no-regret online optimizer \( \mathcal{F} \), at the \( n \)th iteration, we formulate loss \( \ell_n(\pi) \) and feed it to the no-regret optimizer \( \mathcal{F} \). We simply take the output from \( \mathcal{F} \) as the policy \( \pi_{n+1} \) for the next iteration. After \( N \) iterations, we obtain a set of policies \( \{ \pi_1, \ldots, \pi_N \} \). We simply output the best policy \( \tilde{\pi} \in \{ \pi_i \}_{i=1}^N \).

Algorithm [4] provides a practical implementation of the above reduction using Follow-the-Leader [Shalev-Shwartz et al. 2012] as the underlying no-regret online learner. The FTL

**Algorithm 4 N^2R-IL (Non-Interactive No-Regret Imitation Learning)**

1. **Input:** The given MDP, an expert dataset \( D = \{ s_i, a_i, r_i, s_i' \}_{i=1}^M \), with \( s_i \sim d_{\pi^*}, a_i = \pi^*(s_i), r_i = r(s_i, a_i), s_i' \sim P(\cdot|s_i, a_i) \); A classifier \( \mathcal{F} : S \rightarrow A \). Mini-batch size \( K \).
2. Initialize \( \pi_0 \) and a dataset \( D = \emptyset \).
3. for \( n = 0 \) to \( N \) do
4. Sample two mini-batch datasets \( D_1 \) and \( D_2 \) from \( D \)
5. For every tuple \( (s_i, a_i, r_i, s_i') \in D_1 \), roll out \( \pi_n \) from \( s_i \) to estimate \( V^{\pi_n}(s_i) \)
6. For every tuple \( (s_i, a_i, r_i, s_i') \in D_2 \), roll out \( \pi_n \) from \( s_i' \) to estimate \( V^{\pi_n}(s_i') \)
7. Form an estimate of \( A^{\pi_n}(s_i, \pi^*(s_i)) \) as \( \hat{w}_i = r_i + \gamma V^{\pi_n}(s_i') - V^{\pi_n}(s_i) \), \( \forall s_i, s'_i \in D_1 \)
8. Sample \( K \) tuples (i.i.d) from \( D_1 \) with probability proportional to \( \max(0, \hat{w}_i) \) for each tuple \( (s_i, a_i, r_i, s_i') \in D_1 \), and form a new dataset \( D_1' \) with these new \( K \) tuples
9. Data aggregation: \( D = D + D_2 + D_1' \)
10. Call classifier \( \mathcal{F} \) to compute \( \pi_{n+1} = \arg \min_\pi \sum_{(s,a,r,s')} \in D l(\pi(s), a) \)
11. end for

**Reduction to No-Regret Online Learning**

We first introduce some notations. For any policy \( \pi_n \), let us define \( d_{\pi^*, n} \) as the distribution reshaped using the advantage \( A^{\pi^*} \) as the weight:

\[
d_{\pi^*, n}(s) = \frac{1}{\nu_n} d_{\pi^*}(s) \max \left( 0, A^{\pi^*}(s, \pi^*(s)) \right),
\]

where

\[
\nu_n = \sum_i d_{\pi^*}(s_i) \max \left( 0, A^{\pi^*}(s_i, \pi^*(s_i)) \right)
\]

Given \( \beta \in (0, 1) \), let us define a mixed state distribution \( \hat{d}_{\pi^*} \) as:

\[
\hat{d}_{\pi^*}(s) = \beta d_{\pi^*}(s) + (1 - \beta)d_{\pi^*} \cdot \nu_n,
\]

which is a mixed from the expert’s state distribution \( d_{\pi^*} \) and weighted expert’s state distribution \( d_{\pi^*} \). Using \( \hat{d}_{\pi^*} \), let us re-define the loss function \( \ell_n(\pi) \) as:

\[
\ell_n(\pi) = \mathbb{E}_{s \sim \hat{d}_{\pi^*}}[l(\pi(s), \pi^*(s))],
\]

where \( l(a, a') \) is a convex surrogate loss function that upper bounds the zero-one loss \( 1[\pi^*(s) \neq \pi(s)] \) (e.g., \( l(a, a') \) could be hinge loss).

The reduction to no-regret online learning works as follows. With some policy \( \pi_0 \) as the initialization, and with access to any no-regret online optimizer \( \mathcal{F} \), at the \( n \)th iteration, we formulate loss \( \ell_n(\pi) \) and feed it to the no-regret optimizer \( \mathcal{F} \). We simply take the output from \( \mathcal{F} \) as the policy \( \pi_{n+1} \) for the next iteration. After \( N \) iterations, we obtain a set of policies \( \{ \pi_1, \ldots, \pi_N \} \). We simply output the best policy \( \tilde{\pi} \in \{ \pi_i \}_{i=1}^N \).
is implemented using the Data Aggregation (DAgger) trick [Ross et al., 2011]. The algorithm starts with an expert dataset \( D^e \) in the format of \( \{ s_i, a_i, r_i, s'_i \}_{i=1}^M \), where \( s_i \in d_\pi \), \( a_i = \pi^*(s_i), r_i = r(s_i, a_i), s'_i \sim P(\cdot|s_i, a_i) \). Note that in practice such a dataset can be formed by first collecting a set of demonstration trajectories from the expert and then decomposing the samples into the format of \( D^e \). Every iteration, we sample a two mini-batches of data points \( D^i_1 \) and \( D^i_2 \) from the expert dataset. For each tuple \( (s, a, r, s') \), we estimate \( A^\pi_n(s, \pi^*(s)) \) by rolling out \( \pi_n \) at \( s' \), sum over \( r \) and the generated rewards during the roll-out. We then weight each \( s_i \in D^i_1 \) with \( \max(0, A^\pi_n(s_i, a_i)) \). We then sample with replacement from the weighted dataset with probability proportional to the weight \( A^\pi_n(s_i, a_i) \) for each tuple \( (s_i, a_i, r_i, s'_i) \), to form a new dataset \( D_1 \). The mixed distribution \( D^*_{\pi^*, n} \) (with \( \beta = 0.5 \)) then can be approximated by the joint dataset \( D^i_2 + D^i_3 \). Note that at the end of iteration \( n \), the aggregated dataset \( D \) is used to forms an empirical estimation of the aggregated loss so far: \( \sum_{j=1}^n \ell_n(\pi) \), which is optimized by the off-shelf classifier \( F \) by the end of iteration \( n \).

A natural question to ask is what is the performance of \( \pi \) when testing on the underlying MDPs. The following theorem indicates that there exists a policy \( \pi \in \{ \pi_n \}_{n=1}^N \) that is close to the expert policy \( \pi^* \) as. Let \( \epsilon_c = \min_{\pi \in \Pi} \sum_{n=1}^N \ell_n(\pi) \) be the true loss of the best policy in hindsight and \( \delta_N \) be the average regret \( \delta_N = (\sum_{n=1}^N \epsilon_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^N \ell_n(\pi))/N \). Note that \( \delta_N \) approaches to zero as \( N \to \infty \) and the rate depends on the specific underlying convex property of \( \ell_n \) and the specific algorithm one use. If \( \ell_n \) is convex with respect to \( \pi_n \), then \( \delta_n \) decays as \( O(\sqrt{N}/N) \); if \( \ell_n \) is strongly convex, there exists algorithm that can decrease \( \delta_n \) as \( O(\log(N)/N) \) [Hazan et al., 2007; Shalev-Shwartz et al., 2012]. We can quantify the gap between the best policy \( \hat{\pi} \) among \( \{ \pi_n \}_{n=1}^N \) and the expert policy \( \pi^* \) as follows.

**Theorem 6.2.1.** For any sequence of policies \( \{ \pi_n \}_{n=1}^N \) that is generated by running any no-regret online learner on the sequence of loss functions \( \{ \ell_n(\pi) \}_{n=1}^N \), there must exist a policy \( \pi_N \in \{ \pi_n \}_{n=1}^N \) such that:

\[
J(\pi^*) - J(\pi_N) \leq O \left( \frac{(\epsilon_c + \delta_N)^2}{(1 - \gamma)^2} \right).
\]

(6.15)

When \( N \to \infty \), we have:

\[
\lim_{N \to \infty} J(\pi^*) - J(\pi_N) \leq O \left( \frac{\epsilon_c^2}{(1 - \gamma)^2} \right).
\]

In next section, we provide a performance bound for \( \pi \) and we explain how our approach can theoretically outperform previous Imitation Learning\footnote{Note that in this way, the tuples \( (s_i, a_i, r_i, s'_i) \) in \( D^e \) will be correlated. One can reduce correlations by selecting tuple that are multiple step away from each other at every single trajectory.} approaches.

**Analysis and Proof of Theorem 6.2.1**

As \( \{ \pi_i \} \) is generated by a no-regret online optimizer \( F \) running on the sequence of loss functions \( \ell_n \), using the no-regret definition, we must have:

\[
\frac{1}{N} \sum_{n=1}^N \ell_n(\pi_n) - \min_{\pi \in \Pi} \frac{1}{N} \sum_{n=1}^N \ell_n(\pi') = \delta_r \in o(N)/N;
\]

(6.17)
where $\delta_r$ stands for the average regret after $N$ iteration and we know that $\delta_r \to 0$ as $N \to \infty$. Define a average state distribution $\bar{d}_{\pi^*}$ as $\bar{d}_{\pi^*} = (\sum_{n=1}^{N} d_{\pi^*,n})/N$. The second term on the left hand side of the above equation essentially measures the minimum possible classification error we can achieve using the hypothesis class II under $\bar{d}_{\pi^*}$:

$$
\min_{\pi \in \Pi} \frac{1}{N} \sum_{n=1}^{N} \ell_n(\pi) = \min_{\pi \in \Pi} \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [l(\pi(s), \pi^*(s))],
$$

(6.18)

which is a standard multi-class classification (if we have more than two discrete actions) task, and can be optimized by any existing off-shelf multi-class classification oracle. For analysis purpose, we denote $\epsilon_c$ as the risk of the policy $\pi'$ outputted from a black-box classification oracle:

$$
\epsilon_c = \min_{\pi' \in \Pi} \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [l(\pi'(s), \pi^*(s))].
$$

(6.19)

Note that $\epsilon_c$ depends the richness of the policy class $\Pi$. If $\pi^* \in \Pi$, then we can see that $\epsilon_c$ could be zero.

With $\delta_r$ and $\epsilon_c$, we can see that $\sum_{n=1}^{N} \ell_n(\pi_n)/N \leq \delta_r + \epsilon_c$. Hence there must exist a policy $\pi_{i*}$ among $\{\pi_n\}_{n=1}^{N}$ such that:

$$
\mathbb{E}_{s \sim \bar{d}_{\pi_{i*},*}, i} [1[\pi_{i*}(s) \neq \pi^*(s)]] \leq \mathbb{E}_{s \sim \bar{d}_{\pi_{i*},*}, i} [l(\pi(s), \pi^*(s))] \leq \delta_r + \epsilon_c,
$$

(6.20)

where the first inequality comes from the fact that the surrogate loss $l(a, a')$ upper bounds the zero-one loss. For notation simplicity, in the rest of the analysis, we denote $\epsilon = \delta_r + \epsilon_c$.

Using the definition of the mixed distribution $d_{\pi^*, i*}$, we have:

$$
\beta \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [1[\pi(s) \neq \pi^*(s)]] + (1 - \beta) \mathbb{E}_{s \sim \bar{d}_{\pi_{i*},*}} [1[\pi(s) \neq \pi^*(s)]] \leq \epsilon,
$$

(6.21)

which ensures that:

$$
\mathbb{E}_{s \sim \bar{d}_{\pi^*}} [1[\pi(s) \neq \pi^*(s)]] \leq \epsilon/\beta,
$$

(6.22)

$$
\mathbb{E}_{s \sim \bar{d}_{\pi_{i*},*}} [1[\pi_{i*}(s) \neq \pi^*(s)]] \leq \epsilon/(1 - \beta).
$$

(6.23)

Using Performance Difference Lemma, we have:

$$
(1 - \gamma)(J(\pi^*) - J(\pi_{i*})) = \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [A^\pi(s, \pi^*)] \leq \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [\max(0, A^\pi(s, \pi^*))]
$$

$$
= \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [A^\pi(s, \pi^*)] \leq \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [\max(0, A^\pi(s, \pi^*))1[\pi_{i*}(s) \neq \pi^*(s)]]
$$

$$
= \nu_{i*} \mathbb{E}_{s \sim \bar{d}_{\pi_{i*},*}} [1[\pi_{i*}(s) \neq \pi^*(s)]] \leq \frac{\nu_{i*} \epsilon}{1 - \beta}
$$

(6.24)

Now let us bound $\nu_{i*}$. Recall the definition of $\nu_{i*}$:

$$
\nu_{i*} = \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [\max(0, A^{\pi_{i*}}(s, \pi^*(s)))]
$$

$$
= \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [\max(0, A^{\pi_{i*}}(s, \pi^*(s)))1[\pi_{i*}(s) \neq \pi^*(s)]]
$$

$$
\leq V_{\max} \mathbb{E}_{s \sim \bar{d}_{\pi^*}} [1[\pi_{i*}(s) \neq \pi^*(s)]] \leq \frac{V_{\max} \epsilon}{\beta}.
$$

(6.25)

Now go back to Eq. (6.24) replace $\nu_{i*}$ by its upper bound, using the fact that the reward $r(s, a) \in [0, 1]$ and thus $V_{\max} \leq 1/(1 - \gamma)$, we get:

$$
J(\pi^*) - J(\pi_{i*}) \leq \frac{1}{(1 - \gamma)^2 \beta (1 - \beta)} \epsilon^2.
$$

(6.26)
We minimize the RHS of the above inequality by setting $\beta = 0.5$, which gives us:

$$J(\pi^*) - J(\pi^i) \leq \frac{4\epsilon^2}{(1-\gamma)^2} = O\left(\frac{\epsilon^2}{(1-\gamma)^2}\right). \quad (6.27)$$

For future work, we plan to work out the exact expert’s sample complexity for achieving $\delta$-near expert performance, and compare our expert sample complexity to existing theoretical IRL approaches, including supervised classification-based approach [Syed, 2010], min-max game approach [Syed et al. 2008], and margin-based approach [Ratliff et al. 2006].

### 6.3 Proposed Work Three: No-Regret Approximate Policy Iteration

Classic Approximate Policy Iteration performs as follows: at the $n^{th}$ iteration with the latest learned policy $\pi_n$, we compute $\pi_{n+1}$

$$\pi_{n+1} = \arg \min_{\pi \in \Pi} \mathbb{E}_{s \sim \nu} [\mathbb{E}_{s' \sim \pi(s|\cdot)} [A^{\pi_n}(s, a)]] , \quad (6.28)$$

where $\nu$ is some exploration distribution where we can get sample states and $\Pi$ is a pre-defined function class (e.g., neural networks, decision trees). In general such an update does not guarantee that the new policy $\pi_{n+1}$ will be better than $\pi_n$ [Kakade and Langford, 2002, Scherrer, 2014]. To make sure one update procedure results in an improved policy, previous works have focused on conservative update: we make sure that $\pi_{n+1}$ will not be far away from $\pi_n$, where the “distance” between $\pi_{n+1}$ and $\pi_n$ are usually defined using total variation distance [Kakade and Langford, 2002, Schulman et al., 2015a] or KL divergence between $\pi_{n+1}$ and $\pi_n$. People have shown that by forcing $\pi_{n+1}$ to stay near $\pi_n$, we can show that $\pi_{n+1}$ can outperform $\pi_n$. Then, the convergence analysis typically will naturally follow by first showing that how much one can improve per iteration and then show that the algorithm will for sure terminate in finite number of iterations as the objective function’s value is upper bounded. Hence, the central part of designing and analyzing API algorithms is basically coming up new policy iteration procedure that ensures the new policy will outperform the current policy. Most of previous works, such as CPI [Kakade and Langford, 2002], Natural Policy Gradient [Bagnell and Schneider, 2003, Kakade, 2002], and PSDP [Bagnell et al., 2004], all leverage one key idea to achieve this goal: only making small moves when updating $\pi_n$ to $\pi_{n+1}$.

### 6.3.1 Achieving Policy Improvement by No-Regret Update

The natural question we want to ask in this proposed work is that except for heavily constraining the distance between $\pi_{n+1}$ and $\pi_n$, is there other mechanism that we can leverage to learn a new policy $\pi_{n+1}$ that guarantees to outperform $\pi_n$? Well, if we think of $\pi_n$ as an intermediate “expert” that we can mimic, if we recall the performance guarantee of AggreVaTe [Ross and Bagnell, 2014], we immediately get a new approach for policy update: let us run AggreVaTe with $\pi_n$ as an intermediate expert, and let AggreVaTe outputs a new policy $\pi_{n+1}$ that is one-step deviation improvement of $\pi_n$; we then replace the “expert” by $\pi_{n+1}$ and repeat, till AggreVaTe fails to output a policy that is better than the latest “expert”.

49
Per Iteration Improvement

Given policy $\pi_n$ at iteration $n$, if we leverage AggreVaTe to compute $\pi_{n+1}$, how much improvement of $\pi_{n+1}$ over $\pi_n$ we can guarantee? Let us assume that during the update from $\pi_n$ to $\pi_{n+1}$ AggreVaTe performs $K$ steps no-regret update, that generates a sequence of policies $\{\pi_1, ..., \pi_K\}$, and we set $\pi_{n+1} = \arg\min_{i \in \{K\}} J(\pi_i)$. Denote $\bar{\pi}_n$ as the stochastic uniformly mixed policy of $\{\pi_i\}_{i=1}^K$ (i.e., we uniformly pick a policy from $\{\pi_i\}_{i=1}^K$, then execute that policy from the beginning to the end). Recall the guarantee from AggreVaTe, we have:

$$J(\pi_{n+1}) - J(\pi_n) \leq \frac{1}{1 - \gamma} \left( o(K)/K + \min_{\bar{\pi} \in \Pi} E_{s \sim d_{\pi_n}} [A^{\pi_n}(s, \bar{\pi}(s))] \right). \quad (6.29)$$

On the other hand, Conservative Policy Iteration (CPI) \cite{Kakade and Langford 2002} offers a policy improvement in the order of:

$$\frac{\alpha}{1 - \gamma} \min_{\bar{\pi} \in \Pi} E_{s \sim d_{\pi_n}} [A^{\pi_n}(s, \bar{\pi})], \quad (6.30)$$

where $\alpha \in (0, 1)$ is a small real number that is used to ensure a conservative update. Compare the above improvement from CPI to the improvement shown in Eq. \ref{eq:6.29}, we can see that using AggreVaTe we can potentially achieve a larger improvement, as the improvement shown in Eq. \ref{eq:6.29} is not scaled by $\alpha \in (0, 1)$.

Of course, as AggreVaTe itself is an iterative algorithm, to make sure the policy improvement is big, we need to set $K$ to be big enough such that average regret $o(K)/K$ is negligible. However, each inner iteration of AggreVaTe requires to draw samples from the underlying systems. One interesting open question is that in order to guarantee to achieve a fixed amount of policy improvement, is AggreVaTe going to require more samples than say, running CPI repeatedly?

Local Optimality Guarantee

When we cannot find a policy $\pi_{n+1}$ that outperforms $\pi_n$ by much, can we guarantee that $\pi_n$ is already a good policy? The answer to this question is affirmative. We can use similar analysis of CPI \cite{Kakade and Langford 2002} to show that when the right hand side of Inequality \ref{eq:6.29} is upper bounded by $\xi \in \mathbb{R}^+$, then for $\pi_n$, we have:

$$J(\pi_n) - J(\pi^*) \leq \max_s \left( \frac{d_{\pi^*}(s)}{\rho_0(s)} \right) \frac{1}{(1 - \gamma)^2} (\xi - o(K)/K). \quad (6.31)$$

For future work, we plan to implement the iterative-AggreVaTe algorithm and compare it to classic API algorithms such as CPI, PSIDP on multiple simulated robotics control tasks. We are interested in investigating the effect of parameter $K$ (i.e., the number of iteration per AggreVaTe run) on the practical performance.
Chapter 7

Conclusion

Reinforcement Learning is hard. So in this work, we design and analyze several reduction frameworks that reduce the difficult RL problems to easier problems which we understand well, by including additional assumptions and sources of information. We show a general reduction from Bellman Residual Minimization based policy evaluation strategies to no-regret online learning. Such a reduction directly transforms any efficient online learning algorithm to an efficient, provably correct policy evaluation algorithm. To speed up RL, we introduce an expert oracle that we can access and imitate during training. Such an expert oracle can dramatically reduce sample complexity for learning a near-optimal policy. We then relax that assumption that we have access to expert oracles, but instead leverage efficient model-based optimal control black-box solvers to compute intermediate “experts” during training. The general policy iteration framework we proposed—Dual Policy Iteration (DPI), maintains and alternatively updates two policies: a reactive policy is updated in an imitation learning fashion given an “expert” policy, while the “expert” policy is consistently updated by model-based optimal control with a learned local model, under the guidance of the reactive policy.

For future work, we plan to generalize Temporal Difference Learning (TD) to off-policy setting with arbitrary non-linear function approximation. The current TD analysis limits to linear-function approximations and the classic TD algorithm diverges in off-policy setting and even in on-policy setting under non-linear function approximation. The second future direction we are pursuing is to consider the setting that sits between the classic Apprenticeship Learning (e.g., only have a set of expert’s demonstrations before training), and the Imitation Learning setting we considered in this work (e.g., expert oracle is available even during training and we have access to reward signals), a setting where we only have a set of expert demonstrations, but have the ability to reset the system and receive reward signals either during executing a policy or just at the end of execution. The access to reward signal allows us to weight classification example by the advantage of the expert policy over our policy. We have proposed a theoretically sound algorithm and we plan to implement and see if the better theoretical guarantee leads to improved practical performance as well.
Bibliography


Pieter Abbeel, Varun Ganapathi, and Andrew Y Ng. Learning vehicular dynamics, with application to modeling helicopters. In NIPS, pages 1–8, 2005.


