Dual Policy Iteration

Wen Sun¹, Geoffrey J. Gordon¹, Byron Boots², and J. Andrew Bagnell³

¹School of Computer Science, Carnegie Mellon University, USA
²College of Computing, Georgia Institute of Technology, USA
³Aurora Innovation, USA
{wensun, ggordon, dbagnell}@cs.cmu.edu, bboots@cc.gatech.edu

Abstract

A novel class of Approximate Policy Iteration (API) algorithms have recently demonstrated impressive practical performance (e.g., ExIt [2], AlphaGo-Zero [30]). This new family of algorithms maintains, and alternately optimizes, two policies: a fast, reactive policy (e.g., a deep neural network) deployed at test time, and a slow, non-reactive policy (e.g., Tree Search), that can plan multiple steps ahead. The reactive policy is updated under supervision from the non-reactive policy, while the non-reactive policy is improved via guidance from the reactive policy. In this work we study this class of Dual Policy Iteration (DPI) strategy in an alternating optimization framework and provide a convergence analysis that extends existing API theory. We also develop a special instance of this framework which reduces the update of non-reactive policies to model-based optimal control using learned local models, and provides a theoretically sound way of unifying model-free and model-based RL approaches with unknown dynamics. We demonstrate the efficacy of our approach on various continuous control Markov Decision Processes.

1 Introduction

Approximate Policy Iteration (API) [10, 8, 14, 17, 27], including conservative API (CPI) [14], API driven by learned critics [26], or gradient-based API with stochastic policies [9, 6, 13, 28], have played a central role in Reinforcement Learning (RL) for decades and motivated many modern practical RL algorithms. Several existing API methods [8, 14] can provide both local optimality guarantees and global guarantees under strong assumptions regarding the way samples are generated (e.g., access to a reset distribution that is similar to the optimal policy’s state distribution). However, most modern practical API algorithms rely on myopic random exploration (e.g., REINFORCE [37] type policy gradient or \( \epsilon \)-greedy). Sample inefficiency due to random exploration can cause even sophisticated RL methods to perform worse than simple black-box optimization with random search in parameter space [20].

Recently, a new class of API algorithms, which we call Dual Policy Iteration (DPI), has begun to emerge. These algorithms follow a richer strategy 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, used as an expert policy to guide the improvement of the reactive policy. For example, ExIt [2] maintains and updates a UCT-based policy [15] as an intermediate expert. ExIt then updates the reactive policy by directly imitating the tree-based policy which we expect would be better than the reactive policy as it involves a multi-step lookahead search. AlphaGo-Zero [30] employs a similar strategy to achieve super-human performance at the ancient game of Go. The key difference that distinguishes ExIt and AlphaGo-Zero from previous APIs is that they leverage models to perform systematic forward search: the policy resulting from forward search acts as an expert and directly informs the improvement direction for the reactive policy. Hence the
reactive policy improves by imitation instead of trial-and-error reinforcement learning. This strategy
often provides better sample efficiency in practice compared to algorithms that simply rely on locally
random search (e.g., AlphaGo-Zero abandons REINFORCE from AlphaGo [31]).

In this work we provide a general framework for synthesizing and analyzing DPI by considering a
particular alternating optimization strategy with different optimization approaches each forming a
new family of approximate policy iteration methods. We additionally consider the extension to the
RL setting with unknown dynamics. For example, we construct a simple instance of our framework,
where the intermediate expert is computed from Model-Based Optimal Control (MBOC) locally
around the reactive policy, and the reactive policy in turn is updated incrementally under the guidance
of MBOC. The resulting algorithm iteratively learns a local dynamics model, applies MBOC to
compute a locally optimal policy, and then updates the reactive policy by imitation and achieve larger
policy improvement per iteration than classic APIs. The instantiation shares similar spirit from some
previous works from robotics and control literature, including works from [3, 5] and Guided Policy
Search (GPS) [18] (and its variants (e.g., [19, 22, 21])), i.e., using local MBOC to speed up learning
global policies.

To evaluate our approach, we demonstrate our algorithm on discrete MDPs and continuous control
tasks. We show that by integrating local model-based search with learned local dynamics into policy
improvement via an imitation learning-style update, our algorithm is substantially more sample-
efficient than classic API algorithms such as CPI [14], as well as more recent actor-critic baselines
[29], albeit at the cost of slower computation per iteration due to the model-based search. We also
apply the framework to a robust policy optimization setting [7, 4] where the goal is to learn a single
policy that can generalize across environments. In summary, the major practical difference between
DPI and many modern practical RL approaches is that instead of relying on random exploration, the
DPI framework integrates local model learning, local model-based search for advanced exploration,
and an imitation learning-style policy improvement, to improve the policy in a more systematic way.

We also provide a general convergence analysis to support our empirical findings. Although our
analysis is similar to CPI’s, it has a key difference: as long as MBOC succeeds, we can provide a
larger policy improvement than CPI at each iteration. Our analysis is general enough to provide
theoretical intuition for previous successful practical DPI algorithms such as Expert Iteration (ExIt)
[2]. We also analyze how predictive error from a learned local model can mildly affect policy
improvement and show that locally accurate dynamics—a model that accurately predicts next states
under the current policy’s state-action distribution, is enough for improving the current policy. We
believe our analysis of local model predictive error versus local policy improvement can shed light
on further development of model-based RL approaches with learned local models. In summary, DPI
operates in the middle of two extremes: (1) API type methods that update policies locally (e.g.,
first-order methods like policy gradient and CPI), (2) global model-based optimization where one
attempts to learn a global model and perform model-based search. First-order methods have small
policy improvement per iteration and learning a global model displays greater model bias and requires
a dataset that covers the entire state space. DPI instead learns a local model and allows us to integrate
models to leverage the power of model-based optimization to locally improve the reactive policy.

2 Preliminaries

A discounted infinite-horizon Markov Decision Process (MDP) is defined as \((\mathcal{S}, \mathcal{A}, P, c, \rho_0, \gamma)\). Here,
\(\mathcal{S}\) is a set of states, \(\mathcal{A}\) 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. Throughout this paper, we assume that we know
the form of the cost function \(c(s, a)\), but the transition dynamics \(P\) are unknown. We define a stochastic policy
\(\pi\) such that for any state \(s \in \mathcal{S}\), \(\pi(\cdot|s)\) outputs a distribution over action space. The distribution
of states at time step \(t\), induced by running the policy \(\pi\) until and including \(t\), is defined \(\forall s_t:
\begin{align*}
d_\pi^t(s_t) &= \sum_{s_{t-1}, a_{t-1}} \rho_0(s_0) \prod_{i=0}^{t-1} \pi(a_i|s_i) P(s_{i+1}|s_i, a_i),
\end{align*}
where by definition \(d_\pi^0(s) = \rho_0(s)\) for any \(\pi\). The state visitation distribution can be computed \(d_\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t d_\pi^t(s)\). Denote \((d_\pi \pi)\) as the joint state-action distribution such that \(d_\pi \pi(s, a) = d_\pi(s) \pi(a|s)\). We define the value

2
function $V^\pi(s)$, state-action value function $Q^\pi(s,a)$, and the objective function $J(\pi)$ as:

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) | s_0 = s \right], Q^\pi(s,a) = c(s,a) + \gamma \mathbb{E}_{s' \sim P_{s,a}} [V^\pi(s')], J(\pi) = \mathbb{E}_{s \sim p_0} [V^\pi(s)].$$

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 work in the cost setting, in the rest of the paper we refer to $A^\pi$ as the disadvantage function. The goal is to learn a single stationary policy $\pi$ that minimizes $J(\pi)$: $\pi^* = \arg\min_{\pi \in \Pi} J(\pi)$.

For two distributions $P_1$ and $P_2$, $D_{TV}(P_1, P_2)$ denotes 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 $D_{KL}(P_1, P_2) = \int_x P_1(x) \log(P_1(x)/P_2(x)) dx$ denotes the KL divergence.

We introduce Performance Difference lemma (PDL) [14], which will be used extensively in this work:

**Lemma 2.1** For any two policies $\pi$ and $\pi'$, we have: $J(\pi) - J(\pi') = \frac{1}{1 - \gamma} \mathbb{E}_{(s,a) \sim d_\pi} [A^\pi(s,a)]$.

3 Dual Policy Iteration

We propose an alternating optimization framework inspired by the PDL (Lemma 2.1). Consider the min-max optimization framework: $\min_{\pi \in \Pi} \max_{\eta \in \Pi} \mathbb{E}_{s \sim d_{\pi}} [E_{x \sim \pi_x(s)} [A^\eta(x,s,x)]].$ It is not hard to see that the unique Nash equilibrium for the above equation is $(\pi, \eta) = (\pi^*, \pi^*)$. The above min-max proposes a general strategy, which we call Dual Policy Iteration (DPI): alternatively fix one policy and update the second policy. Mapping to previous practical DPI algorithms [2, 30], $\pi$ stands for the fast reactive policy and $\eta$ corresponds to the tree search policy. For notation purposes, we use $\pi_n$ and $\eta_n$ to represent the two policies in the $n$th iteration. Below we introduce one instance of DPI for settings with unknown models (hence no tree search), first describe how to compute $\eta_n$ from a given reactive policy $\pi_n$ (Sec. 3.1), and then describe how to update $\pi_n$ to $\pi_{n+1}$ via imitating $\eta_n$ (Sec. 3.2).

3.1 Updating $\eta$ with MBOC using Learned Local Models

Given $\pi_n$, the objective function for $\eta$ becomes: $\max_{\eta} \mathbb{E}_{s \sim d_{\pi_n}} [E_{x \sim \pi_x(s)} [A^\eta(x,s,x)]]$. From PDL we can see that updating $\eta$ is equivalent to finding the optimal policy $\pi^*$: $\arg \max_{\eta} (J(\pi_n) - J(\eta)) = \arg \min_{\eta} J(\eta)$, regardless of what $\pi_n$ is. As directly minimizing $J(\eta)$ is as hard as the original problem, we update $\eta$ locally by constraining it to a trust region around $\pi_n$:

$$\arg \min_{\eta} J(\eta), \text{ s.t., } \mathbb{E}_{s \sim d_{\pi_n}} D_{TV}([\eta(|s), \pi_n(|s)]) \leq \alpha. \hspace{1cm} (1)$$

To solve the constraint optimization problem in Eq 1, we propose to learn $P_{s,a}$ and use it with any off-the-shelf model-based optimal control algorithm. Moreover, thanks to the trust region, we can simply learn a local dynamics model, under the state-action distribution $d_{\pi_n} \pi_n$. We denote the optimal solution to the above constrained optimization (Eq. 1) under the real model $P_{s,a}$ as $\eta_n^\pi$. Note that, due to the definition of the optimality, $\eta_n^\pi$ must perform better than $\pi_n$: $J(\pi_n) - J(\eta_n^\pi) \geq \Delta_n(\alpha)$, where $\Delta_n(\alpha) \geq 0$ is the performance gain from $\eta_n^\pi$ over $\pi_n$. When the 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 $\mathbb{E}_{(s,a) \sim d_{\pi_n} \pi_n} D_{TV}(P_{s,a}, \hat{P}_{s,a})$ is small. Optimizing $D_{TV}$ directly is hard, but note that, by Pinsker’s inequality, we have $D_{KL}(P_{s,a}, \hat{P}_{s,a}) \geq D_{TV}(P_{s,a}, \hat{P}_{s,a})^2$, which indicates that we can optimize a surrogate loss defined by a KL-divergence:

$$\arg \min_{P \in \mathcal{P}} \mathbb{E}_{s \sim d_{\pi_n} \pi_n} D_{KL}(P_{s,a}, \hat{P}_{s,a}) = \arg \min_{P \in \mathcal{P}} \mathbb{E}_{s \sim d_{\pi_n} \pi_n} [\log \hat{P}_{s,a}(s')] \hspace{1cm} (2)$$

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., $P_{s,a}$). As we will show later, to ensure policy improvement, we just need a learned model to perform well under $d_{\pi_n} \pi_n$ (i.e., no training and
testing distribution mismatch as one will have for global model learning). For later analysis purposes, we denote $P$ as the MLE in Eq. 2 and assume $P$ is $\delta$-optimal under $d_{\pi_n, \hat{\pi}_n}$:

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

(3)

where $\delta \in \mathbb{R}^+$ is controlled by the complexity of model class $P$ and by the amount of training data we sample using $\pi_n$, which can be analyzed by standard supervised learning theory. After achieving a locally accurate model $\hat{P}$, we solve Eq. 1 using any existing stochastic MBOC solvers. Assume a MBOC solver returns an optimal policy $\eta_n$ under the estimated model $\hat{P}$ subject to trust-region:

$$\eta_n = \arg \min_{\pi} J(\pi), \text{ s.t., } s_{t+1} \sim \hat{P}_{s_t,a_t}, \mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi, \eta_n) \leq \alpha.$$

(4)

At this point, a natural question is: If $\eta_n$ is solved by an MBOC solver under $\hat{P}$, by how much can $\eta_n$ outperform $\pi_n$ when executed under the real dynamics $P$? Recall that the performance gap between the real optimal solution $\eta_n^*$(optimal under $P$) and $\pi_n$ is denoted as $\Delta_n(\alpha)$. The following theorem quantifies the performance gap between $\eta_n$ and $\pi_n$ using the learned local model’s predictive error $\delta$:

**Theorem 3.1** Assume $\hat{P}_{s,a}$ satisfies Eq. 3, and $\eta_n$ is the output of a MBOC solver for the optimization problem defined in Eq. 4, then we have:

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

The proof of the above theorem can be found in Appendix A.2. Theorem 3.1 indicates that when the model is locally accurate, i.e., $\delta$ is small (e.g., $P$ is rich and we have enough data from $d_{\pi_n, \pi_n}$), $\alpha$ 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 learned model $\hat{P}$ finds a nearly local-optimal solution $\eta_n$ that outperforms $\pi_n$. With a better $\eta_n$, now we are ready to improve $\pi_n$ via imitating $\eta_n$.

### 3.2 Updating $\pi$ via Imitating $\eta$

Given $\eta_n$, we compute $\pi_{n+1}$ by performing the following constrained optimization procedure:

$$\arg \min_{\pi} \mathbb{E}_{s \sim d_{\pi_n}} \left[\mathbb{E}_{a \sim \pi(\cdot|s)} [A^{\eta_n}(s, a)]\right], \text{ s.t., } \mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi(\cdot|s), \pi_n(\cdot|s)) \leq \beta.$$

(5)

Note that the key difference between Eq. 5 and classic API policy improvement procedure is that we use $\eta_n$’s disadvantage function $A^{\eta_n}$, i.e., we are performing imitation learning by treating $\eta_n$ as an expert in this iteration [24, 33]. We can solve Eq. 5 by converting it to supervised learning problem such as cost-sensitive classification [14] by sampling states and actions from $\pi_n$ and estimating $A^{\eta_n}$ via rolling out $\eta_n$, subject to an L1 constraint.

Note that a CPI-like update approximately solves the above constrained problem as well:

$$\pi_{n+1} = (1 - \beta)\pi_n + \beta \pi_n^*, \text{ where } \pi_n^* = \arg \min_{\pi} \mathbb{E}_{s \sim d_{\pi_n}} \left[\mathbb{E}_{a \sim \pi(\cdot|s)} [A^{\eta_n}(s, a)]\right].$$

(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, the update in Eq. 6 can be understood as first solving the objective function to obtain $\pi_n^*$ without considering the constraint, and then moving $\pi_n$ towards $\pi_n^*$ until the boundary of the constraint is reached.

### 3.3 DPI: Combining Updates on $\pi$ and $\eta$

In summary, assume MBOC is used for Eq. 1, DPI operates in an iterative way: with $\pi_n$:

1. Fit MLE $\hat{P}$ on states and actions from $d_{\pi_n, \pi_n}$ (Eq. 2).
2. $\eta_n \leftarrow$ MBOC($\hat{P}$), subject to trust region $\mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi, \eta_n) \leq \alpha$ (Eq. 4).
3. Update to $\pi_{n+1}$ by imitating $\eta_n$, subject to trust region $\mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi, \pi_n) \leq \beta$ (Eq. 5).

The above framework shows how $\pi$ and $\eta$ are tightened together to guide each other’s improvements: the first step corresponds classic MLE under $\pi_n$’s state-action distribution: $d_{\pi_n, \pi_n}$; the second step corresponds to model-based policy search around $\pi_n$ ($\hat{P}$ is only locally accurate); the third step corresponds to updating $\pi$ by imitating $\eta$ (i.e., imitation). Note that in practice MBOC solver (e.g., a second order optimization method, as we will show in our practical algorithm below) could be computationally expensive and slow (e.g. tree search in ExIt and AlphaGo-Zero), but once $\hat{P}$ is provided, MBOC does not require additional samples from the real system.
Connections to Previous works We can see that the above framework generalizes several previous work from API and IL. (a) If we set $\alpha = 0$ in the limit, we reveal CPI (assuming we optimize with Eq. 6), i.e., no attempt to search for a better policy using model-based optimization. (b) 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, and MBOC plays the role of forward search. (c) when an optimal expert policy $\pi^*$ is available during and only during training, we can set every $\eta_n$ to be $\pi^*$, and DPI then reveals a previous IL algorithm—AGGREGATE [33].

4 Analysis of Policy Improvement

We provide a general convergence analysis for DPI. The trust region constraints in Eq. 1 and Eq. 5 tightly combines MBOC and policy improvement together, and is the key to ensure monotonic improvement and achieve larger policy improvement per iteration than existing APIs.

Define $\Delta_n(\pi_{n+1})$ as the disadvantage of $\pi_{n+1}$ over $\pi_n$ under $d_\pi$: $\Delta_n(\pi_{n+1}) = \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{E}_{a \sim \pi_{n+1}(\cdot|s)} [A^m(s, a)] \right]$. Note that $\Delta_n(\pi_{n+1})$ is at least non-negative (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 $\Delta_n(\pi_{n+1}) = 0$, which means we can hope that the IL procedure (Eq. 5) finds a policy $\pi_{n+1}$ that achieves $\Delta_n(\pi_{n+1}) < 0$ (i.e., local improvement over $\eta_n$). 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. 1 and Eq. 5. Following Theorem 4.1 from [14], we define $\varepsilon = \max_s \left[ \mathbb{E}_{a \sim \pi_{n+1}(\cdot|s)} [A^m(s, a)] \right]$, which measures the maximum possible one-step improvement one can achieve from $\eta_n$. The following theorem states the performance improvement:

**Theorem 4.1** Solve Eq. 1 to get $\eta_n$ and Eq. 5 to get $\pi_{n+1}$. The improvement of $\pi_{n+1}$ over $\pi_n$ is:

$$J(\pi_{n+1}) - J(\pi_n) \leq \frac{\beta \varepsilon}{(1 - \gamma)^2} - \frac{\Delta_n(\pi_{n+1})}{1 - \gamma} \leq \frac{\Delta_n(\pi_n)}{1 - \gamma}. \quad (7)$$

The proof of Theorem 4.1 is provided in Appendix A.3. When $\beta$ is small, we are guaranteed to find a policy $\pi_{n+1}$ where the total cost decreases by $\Delta_n(\alpha) + |\Delta_n(\pi_{n+1})|/(1 - \gamma)$ compared to $\pi_n$. Note that classic CPI’s per iteration improvement [14, 28] only contains a term that has the similar meaning and magnitude of the second term in the RHS of Eq. 7. Hence DPI can improve the performance of CPI by introducing an extra term $\Delta_n(\alpha)$, and the improvement could be substantial when there exists a locally optimal policy $\eta_n$ that is much better than the current reactive policy $\pi_n$. Such $\Delta_n(\alpha)$ comes from the explicit introduction of a model-based search into the training loop, which does not exist in classic APIs. From a practical point view, modern MBOCs are usually second-order methods, while APIs are usually first-order (e.g., REINFORCE and CPI). Hence it is reasonable to expect $\Delta_n(\alpha)$ itself will be larger than API’s policy improvement per iteration. Connecting back to ExIt and AlphaGo-Zero under model-based setting, $\Delta_n(\alpha)$ stands for the improvement of the tree-based policy over the current deep net reactive policy. In ExIt and AlphaGo Zero, the tree-based policy $\eta_n$ performs fixed depth forward search followed by rolling out $\pi_n$ (i.e., bottom up by $V^\pi_n(s)$), which ensures the expert $\eta_n$ outperforms $\pi_n$.

When $|\Delta_n(\alpha)|$ and $|\Delta_n(\pi_{n+1})|$ are small, i.e., $|\Delta_n(\alpha)| \leq \xi$ and $|\Delta_n(\pi_{n+1})| \leq \xi$, then we can guarantee that $\eta_n$ and $\pi_n$ are good policies, under the stronger assumption that the initial distribution $\rho_0$ happens to be a good distribution (e.g., close to $d_{\pi^*}$), and the realizable assumption: $\min_{\pi \in \Pi} \mathbb{E}_{a \sim d_{\pi}} [\mathbb{E}_{s \sim \pi \sim (\cdot|s)} [A^m(s, a)]] = \mathbb{E}_{s \sim d_{\pi}} [\min_{a \sim A} [A^m(s, a)]]$ holds. We show in Appendix A.4 that under the realizable assumption:

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

The term $(\max_s(d_{\pi^*}(s)/\rho_0(s)))$ measures the distribution mismatch between the initial distribution $\rho_0$ and the optimal policy $\pi^*$, and appears in some previous API algorithms—CPI [14] and PSDP [8]. $A_0$ that is closer to $d_{\pi^*}$ (e.g., let experts reset the agent’s initial position if possible) ensures better global performance guarantee. CPI considers a setting where a good reset distribution $\nu$ (different from $\rho_0$) is available, DPI can leverage such reset distribution by replacing $\rho_0$ by $\nu$ at training.

In summary, we can expect larger per-iteration policy improvement from DPI compared to CPI (and TRPO which has similar per iteration policy improvement as CPI), thanks to the introduction of local model-based search. The final performance bound of the learned policy is in par with CPI and PSDP.
5 An Instance of DPI

In this section, we dive into the details of each update step of DPI and suggest one practical instance of DPI, which can be used in continuous control settings. We denote $T$ as the maximum possible horizon.\(^1\) We denote the state space $S \subseteq \mathbb{R}^{d_s}$ and action space $A \subseteq \mathbb{R}^{d_a}$. 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 $\eta = \{\eta_t\}_{0 \leq t \leq T}$, where $\eta_t(\cdot|s) = N(K_t s + k_t, P_t)$ with control gain $K_t \in \mathbb{R}^{d_s \times d_a}$, bias term $k_t \in \mathbb{R}^{d_a}$ and Covariance $P_t \in \mathbb{R}^{d_a \times d_a}$. We will use $\Theta = \{K_t, k_t, P_t\}_{0 \leq t \leq T}$ 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 $D_{TV}(\pi_n, \pi)$ (note total variation distance is symmetric) with the KL-divergence $D_{KL}(\pi_n, \pi)$, which allows us to leverage Natural Gradient $[13, 6]$. To summarize, $\pi_n$ and $\eta_n$ are short for $\pi_{\theta_n}$ and $\eta_{\theta_n} = \{N(K_t s + k_t, P_t)\}_t$, respectively. Below we first describe how to compute $\eta_{\theta_n}$ given $\pi_n$ (Sec. 5.1), and then describe how to update $\pi$ via imitating $\eta_{\theta_n}$ using Natural Gradient (Sec. 5.2).

5.1 Updating $\eta_{\theta}$ with MBOC using Learned Time Varying Linear Models

We explain here how to find $\eta_n$ given $\pi_n$ using MBOC. In our implementation, we use Linear Quadratic Gaussian (LQG) optimal control $[16]$ as the black-box optimal control solver. We learn a sequence of time varying linear Gaussian transition models to represent $\bar{P}: \forall t \in [1, T]$, 

$$s_{t+1} \sim N(A_t s_t + B_t a_t + c_t, \Sigma_t), \quad (8)$$

where $A_t, B_t, c_t, \Sigma_t$ 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 dynamics $P(s, a)$ may be complicated over the entire space, linear dynamics could locally approximate the dynamics well (after all, our theorem only requires $\bar{P}$ to have low predictive error under $d_{\pi_n, \pi_n}$).

Next, to find a locally optimal policy under linear-Gaussian transitions (i.e., Eq. 4), we add the KL constraint to the objective with Lagrange multiplier $\mu$ and form an equivalent min-max problem:

$$\min_{\eta} \max_{\mu \geq 0} \mathbb{E} \left[ \sum_{t=1}^{T} \gamma^{t-1} c(s_t, a_t) \right] + \mu \left( \sum_{t=1}^{T} \gamma^{t-1} \mathbb{E}_{s \sim d_{\eta}} [D_{KL}(\eta, \pi_n)] - \alpha \right), \quad (9)$$

where $\mu$ is the Lagrange multiplier, which can be solved by alternatively updating $\eta$ and $\mu$ $[18]$. For a fixed $\mu$, using the derivation from $[18]$, ignoring terms that do not depend on $\eta$, Eq. 9 can be written:

$$\arg \min_{\eta} \mathbb{E} \left[ \sum_{t=1}^{T} \gamma^{t-1} c(s_t, a_t) - \mu - \log \pi_n(a_t|s_t) \right] - \sum_{t=0}^{T} \gamma^{t-1} \mathbb{E}_{s \sim d_{\eta}} [\mathcal{H}(\eta(\cdot|s))], \quad (10)$$

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: $c'(s_t, a_t) = c(s_t, a_t) - \mu - \log(\pi_n(a_t|s_t))$, and an entropy regularization on $\pi$. 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. 10 can be found exactly by a Dynamic Programming (DP) based approach, the Linear Quadratic Regulator (LQR) $[16]$. Given a dataset $\{(s_t, a_t), c'(s_t, a_t)\}$ collected while executing $\pi_n$, we can fit a quadratic approximation of $c'(s, a)$ $[39, 18]$. With a quadratic approximation of $c'$ and linear dynamics, we solve Eq. 10 for $\eta$ exactly by LQR $[39]$. Once we get $\eta$, we go back to Eq. 9 and update the Lagrange multiplier $\mu$, for example, by projected gradient ascent $[40]$. Upon convergence, LQR gives us a sequence of time-dependent linear Gaussian policies together with a sequence of analytic quadratic cost-to-go functions $Q_{\eta}(s, a)$, and quadratic disadvantage functions $A_{\eta}^{\mu_n}(s, a)$, for all $t \in [T]$.

\(^1\)Note $T$ is the maximum possible horizon which could be long. Hence, we still want to output a single policy, especially when the policy is parameterized by complicated non-linear function approximators like deep nets.

\(^2\)Small $D_{KL}$ leads to small $D_{TV}$, as by Pinsker’s inequality, $D_{KL}(q, p)$ (and $D_{KL}(p, q)$) $\geq D_{TV}(p, q)$. 

| Algorithm 1 AGGREvATeD-GPS |
|-----------------------------|
| 1: **Input:** Parameters $\alpha \in \mathbb{R}^+$, $\beta \in \mathbb{R}^+$. |
| 2: **Initialized** $\pi_{\theta_0}$. |
| 3: for $n = 0$ to ... do |
| 4: Execute $\pi_{\theta_n}$ to generate a set of trajectories |
| 5: Fit local linear dynamics $\hat{P}$ (Eq. 8) using $\{s_t, a_t, s_{t+1}\}$ collected from step 1 |
| 6: Solve the minmax in Eq. 9 subject to $\hat{P}$ to obtain $\eta_{\theta_n}$ and form disadvantage $A^{\mu_n}$. |
| 7: Compute $\theta_{n+1}$ by NGD (Eq. 12) |
| 8: end for |


5.2 Updating $\pi_\theta$ via imitating $\eta_\theta$ using Natural Gradient

Performing a second order Taylor expansion of the KL constraint $\mathbb{E}_{s \sim d_{\pi_n}} [D_{KL}(\pi_n(\cdot|a), \pi(\cdot|s; \theta))]$ around $\theta_n$ [13, 6], we get the following constrained optimization problem:

$$
\min_{\theta} \mathbb{E}_{s \sim d_{\pi_n}} [\mathbb{E}_{a \sim \pi(\cdot|s; \theta)} [A^\theta_{\eta_\theta n}(s, a)]] \text{ s.t. } (\theta - \theta_n)^T F_{\theta_n} (\theta - \theta_n) \leq \beta, \tag{11}
$$

where $F_{\theta_n}$ is the Hessian of the KL constraint $\mathbb{E}_{s \sim d_{\pi_n}} D_{KL}(\pi_{\theta_n}, \pi_{\theta})$ (i.e., Fisher information matrix), measured at $\theta_n$. Denote the objective (i.e., the first term in Eq. 11) as $L_n(\theta)$, and denote $\nabla_{\theta_n} \pi_{\theta_n}$ as $\nabla_{\theta_n} L_n(\theta)|_{\theta=\theta_n}$. we can optimize $\theta$ by performing natural gradient descent (NGD):

$$
\theta_{n+1} = \theta_n - \mu F_{\theta_n}^{-1} \nabla_{\theta_n} L_n(\theta)|_{\theta=\theta_n}, \quad \mu = \sqrt{\beta/(\nabla_{\theta_n}^T F_{\theta_n}^{-1} \nabla_{\theta_n})}. \tag{12}
$$

The specific $\mu$ above ensures the KL constraint is satisfied. More details about the imitation update on $\pi$ can be found in Appendix B.3.

Summary If we consider $\eta$ as an expert, NGD is similar to natural gradient AGGRESSIVATED—a differential IL approach [33]. We summarizes the procedures presented in Sec. 3.1&5.2 in Alg. 1, which we name as AGGRESSIVATED-GPS, stands for the fact that we are using MBOC to Guide Policy Search [18, 22] via AGGRESSIVATED-type update. Every iteration, we run $\pi_{\theta_n}$ on $P$ to gather samples. We estimate time dependent local linear dynamics $P$ and then leverage an OC solver (e.g. LQR) to solve the Lagrangian in Eq. 9 to compute $\eta_{\theta_n}$ and $A^{\eta_{\theta_n}}$. We then perform NGD to update to $\pi_{n+1}$.

5.3 Additional Related Works

The most closely related work with respect to Alg. 1 is Guided Policy Search (GPS) for unknown dynamics [18] and its variants (e.g., [19, 22, 21]). GPS (including its variants) demonstrates model-based optimal control approaches can be used to speed up training policies parameterized by rich non-linear function approximators (e.g., deep networks) in large-scale applications. While Alg. 1 in high level follows GPS’s iterative procedure of alternating reactive policy improvement and MBOC, the main difference between Alg. 1 and GPSs are the update procedure of the reactive policy. Classic GPS, including the mirror descent version, phrases the update procedure of the reactive policy as a behavior cloning procedure, i.e., given an expert policy $\eta$, we perform $\min_{\pi} D_{KL}(d_{\eta}|d_{\pi})$.

Note that our approach to updating $\pi$ is fundamentally on-policy, i.e., we generate samples from $\pi$. Moreover, we update $\pi$ by performing policy iteration against $\eta$, i.e., $\pi$ approximately acts greedily with respect to $A^\pi$, which resulting a key difference: if we limit the power of MBOC, i.e., set the trust region size in MBOC step to zero in both DPI and GPS, then our approach reduces to CPI and thus improves $\pi$ to local optimality. GPS and its variants, by contrast, have no ability to improve the reactive policy in that setting.

6 Experiments

We tested our approach on several MDPs: (1) a set of random discrete MDPs (Garnet problems [27]) (2) Cartpole balancing [34], (3) Helicopter Aerobatics (Hover and Funnel) [1], (4) Swimmer, Hopper and Half-Cheetah from the MuJoCo physics simulator [35]. The goals of these experiments are: (a) to experimentally verify that using $A^\pi$ from the intermediate expert $\eta$ computed by model-based search to perform policy improvement is more sample-efficient than using $A^\eta$. (b) to show that our approach can be applied to robust policy search and can outperform existing approaches [4].

6.1 Comparison to CPI on Discrete MDPs

Following [27], we randomly create ten discrete MDPs with 1000 states and 5 actions. Different from the techniques we introduced in Sec. 5.2 for continuous settings, here we use the conservative

---

3See Line 3 in Alg.2 in [22], where in principle a behavior cloning on $\pi$ uses samples from expert $\eta$ (i.e., off-policy samples). We note, however, in actual implementation some variants of GPS tend to swap the order of $\pi$ and $\eta$ inside the KL, often resulting a on-policy sampling strategy (e.g.,[21]). We also note a Mirror Descent interpretation and analysis to explain GPS’s convergence [22] implies the correct way to perform a projection is to minimize the reverse KL, i.e., $\arg \min_{\pi \in \Pi} D_{KL}(d_{\pi}|d_{\eta})$. This in turn matches the DPI intuition: one should attempt to find a policy $\pi$ that is similar to $\eta$ under the state distribution of $\pi$ itself.
update shown in Eq. 6 to update the reactive policy, where each $\pi^*_n$ is a linear classifier and is trained using regression-based cost-sensitive classification on samples from $d_n$. The feature for each state $\phi(s)$ is a 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 (hence expert $\eta$ and reactive policy $\pi$ have different feature representation) and is computed using exact VI under $\hat{P}$ and $c'(s,a)$ (hence we name our approach here as AGGREGATE-VI). The setup and the conservative update implementation is detailed in Appendix B.1. Fig. 1 reports the statistical performance of our approach and CPI over the 10 random discrete MDPs. Note that our approach is more sample-efficient than CPI, although we observed it is slower than CPI per iteration as we ran VI using learned model. We tune $\beta$ and neither CPI nor our approach uses line search on $\beta$. The major difference between AGGREGATE-VI and CPI here is that we used $A\eta$ instead of $A\pi$ to update $\pi$.

### 6.2 Comparison to Actor-Critic in Continuous Settings

We compare against TRPO-GAE [29] on a set of continuous control tasks. The setup is detailed in Appendix B.4. 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 LQR as the underlying MBOC solver and we name our approach as AGGREGATE-LQR. Fig. 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 than TRPO-GAE albeit slower per iteration as we perform MBOC. 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 learns $A\pi$ using rollouts. Overall, our approach converges faster than TRPO-GAE (i.e., uses less samples), which again indicates the benefit of using $A\eta$ in policy improvement.

### 6.3 Application on Robust Policy Optimization

One application for our approach is robust policy optimization [38], 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 reactive 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 [7, 4], as the training environments together act as “regularization” to avoid overfitting and provide generalization.
More formally, let us assume that we have $M$ training environments. At iteration $n$ with $\pi_{\theta_n}$, we execute $\pi_{\theta_n}$ on the $i$'th environment, generate samples, fit local models, and call MBOC associated with the $i$'th environment to compute $\eta_{\Theta_i}$, for all $i \in [M]$. With $A^{\text{best}}_n$, for all $i \in [M]$, we consider all training environments equally and formalize the objective $L_n(\theta)$ as

$$L_n(\theta) = \sum_{i=1}^{M} E_{s \sim d_{\pi_n}} [E_{a \sim \pi(\cdot | s; \theta)} [A^{\eta_{\Theta_i}}]]$$

We update $\theta_n$ to $\theta_{n+1}$ by NGD on $L_n(\theta)$. Intuitively, we update $\pi_\theta$ by imitating $\eta_{\Theta_i}$ simultaneously for all $i \in [M]$.

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 use 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 [4]. We also ran our algorithm on a single randomly picked training environment, but still tested on test environments, which is denoted as non-robust in Fig. 2. Fig. 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 fails to further converge, which illustrates the overfitting phenomenon: the learned policy overfits to one particular training environment.

7 Conclusion

We present and analyze Dual Policy Iteration—a framework that alternatively computes a non-reactive policy via more advanced and systematic search, and updates a reactive policy via imitating the non-reactive one. Recent algorithms that have been successful in practice, like AlphaGo-Zero and ExIt, are subsumed by the DPI framework. We then provide a simple instance of DPI for RL with unknown dynamics, where the instance integrates local model fit, local model-based search, and reactive policy improvement via imitating the teacher—the nearly local-optimal policy resulting from model-based search. We theoretically show that integrating model-based search and imitation into policy improvement could result in larger policy improvement at each step. We also experimentally demonstrate the improved sample efficiency compared to strong baselines.

Our work also opens some new problems. In theory, the performance improvement during one call of optimal control with the local accurate model depends on a term that scales quadratically with respect to the horizon $1/(1 - \gamma)$. We believe the dependency on horizon can be brought down by leveraging system identification methods focusing on multi-step prediction [36, 32]. On the practical side, our specific implementation has some limitations due to the choice of LQG as the underlying OC algorithm. LQG-based methods usually require the dynamics and cost functions to be somewhat smooth so that they can be locally approximated by polynomials. We also found that LQG planning horizons must be relatively short, as the approximation error from polynomials will likely compound over the horizon. We plan to explore the possibility of learning a non-linear dynamics and using more advanced non-linear optimal control techniques such as Model Predictive Control (MPC) for more sophisticated control tasks.

Acknowledgement

We thank Sergey Levine for valuable discussion. WS is supported in part by Office of Naval Research contract N0001415112365
References

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

[2] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

[3] Christopher G Atkeson. Using local trajectory optimizers to speed up global optimization in dynamic programming. In *Advances in neural information processing systems*, pages 663–670, 1994.

[4] Christopher G Atkeson. Efficient robust policy optimization. In *American Control Conference (ACC)*, 2012, pages 5220–5227. IEEE, 2012.

[5] Christopher G Atkeson and Jun Morimoto. Nonparametric representation of policies and value functions: A trajectory-based approach. In *Advances in neural information processing systems*, pages 1643–1650, 2003.

[6] J Andrew Bagnell and Jeff Schneider. Covariant policy search. *IJCAI*, 2003.

[7] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA*. *IEEE International Conference on*, volume 2, pages 1615–1620. IEEE, 2001.

[8] J Andrew Bagnell, Sham M Kakade, Jeff G Schneider, and Andrew Y Ng. Policy search by dynamic programming. In *Advances in neural information processing systems*, pages 831–838, 2004.

[9] Jonathan Baxter and Peter L Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[10] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995.*, *Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[11] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine. Guided policy search code implementation, 2016. URL http://rll.berkeley.edu/gps. Software available from rll.berkeley.edu/gps.

[12] Alex Gorodetsky, Sertac Karaman, and Youssef Marzouk. Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015. doi: 10.15607/RSS.2015.XI.015.

[13] Sham Kakade. A natural policy gradient. *NIPS*, 2002.

[14] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[15] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[16] Huibert Kwakernaak and Raphael Sivan. *Linear optimal control systems*, volume 1. Wiley-Interscience New York, 1972.

[17] Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Analysis of a classification-based policy iteration algorithm. In *ICML-27th International Conference on Machine Learning*, pages 607–614. Omnipress, 2010.

[18] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[19] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
[20] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. \textit{arXiv preprint arXiv:1803.07055}, 2018.

[21] William Montgomery, Anurag Ajay, Chelsea Finn, Pieter Abbeel, and Sergey Levine. Reset-free guided policy search: efficient deep reinforcement learning with stochastic initial states. In \textit{Robotics and Automation (ICRA), 2017 IEEE International Conference on}, pages 3373–3380. IEEE, 2017.

[22] William H Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In \textit{Advances in Neural Information Processing Systems}, pages 4008–4016, 2016.

[23] Stephane Ross and Drew Bagnell. Agnostic system identification for model-based reinforcement learning. In \textit{Proceedings of the 29th International Conference on Machine Learning (ICML-12)}, pages 1703–1710, 2012.

[24] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. \textit{arXiv preprint arXiv:1406.5979}, 2014.

[25] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In \textit{AISTATS}, 2011.

[26] Gavin A Rummery and Mahesan Niranj. \textit{On-line Q-learning using connectionist systems}, volume 37. University of Cambridge, Department of Engineering, 1994.

[27] Bruno Scherrer. Approximate policy iteration schemes: a comparison. In \textit{International Conference on Machine Learning}, pages 1314–1322, 2014.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In \textit{ICML}, pages 1889–1897, 2015.

[29] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. \textit{arXiv preprint arXiv:1506.02438}, 2015.

[30] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. \textit{Nature}, 550(7676):354, 2017.

[31] David Silver et al. Mastering the game of go with deep neural networks and tree search. \textit{Nature}, 2016.

[32] Wen Sun, Arun Venkatraman, Byron Boots, and J Andrew Bagnell. Learning to filter with predictive state inference machines. In \textit{ICML}, 2016.

[33] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggregated: Differentiable imitation learning for sequential prediction. \textit{ICML}, 2017.

[34] Richard S Sutton and Andrew G Barto. \textit{Introduction to reinforcement learning}, volume 135. MIT Press Cambridge, 1998.

[35] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In \textit{Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on}, pages 5026–5033. IEEE, 2012.

[36] Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. \textit{AAAI}, 2015.

[37] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. \textit{Machine learning}, 1992.

[38] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. \textit{Robust and optimal control}, volume 40. Prentice hall New Jersey, 1996.

[39] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy, 2010.

[40] Martin Zinkevich. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In \textit{ICML}, 2003.
A Missing Proofs

A.1 Useful Lemmas

As we work in finite probability space, we will use the following fact regarding total variation distance and L1 distance for any two probability measures $P$ and $Q$:

$$\|P - Q\|_1 = 2D_{TV}(P, Q).$$  \hfill (13)

Recall that $d_\gamma = (1 - \gamma)\sum_{t=0}^{\infty} \gamma^t d_t$. The following lemma shows that if two policies are close with each other in terms of the trust region constraint we defined in the paper, then the state visitations of the two policies are not that far away.

**Lemma A.1** Given any two policy $\pi_1$ and $\pi_2$ such that $E_{s \sim d_{\pi_1}}[D_{TV}(\pi_1(\cdot|s), \pi_2(\cdot|s))] \leq \alpha$, then we have:

$$\|d_{\pi_1} - d_{\pi_2}\|_1 \leq \frac{2\alpha \gamma}{1 - \gamma}.$$  \hfill (14)

**Proof:** Fix a state $s$ and time step $t$, let us first consider $d^t_{\pi_1}(s) - d^t_{\pi_2}(s)$.

$$d^t_{\pi_1}(s) - d^t_{\pi_2}(s) = \sum_{s_0, a_0} \sum_{a_1, a_2, \ldots, a_{t-1}} (\rho(s_0)\pi_1(a_0|s_0)P_{s_0, a_0}(s_2)\ldots\pi_1(a_{t-1}|s_{t-1})P_{s_{t-1}, a_{t-1}}(s)$$

$$- \rho(s_0)\pi_2(a_0|s_0)P_{s_0, a_0}(s_1)\ldots\pi_2(a_{t-1}|s_{t-1})P_{s_{t-1}, a_{t-1}}(s)$$

$$= \sum_{s_0, a_0} \pi_1(a_0|s_0)P_{s_0, a_0}(s_1)\ldots\pi_1(a_{t-1}|s_{t-1})P_{s_{t-1}, a_{t-1}}(s) - \sum_{s_0, a_0} \pi_2(a_0|s_0)P_{s_0, a_0}(s_1)\ldots\pi_2(a_{t-1}|s_{t-1})P_{s_{t-1}, a_{t-1}}(s)$$

$$= \sum_{s_0, a_0} \pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_1) - \sum_{s_0, a_0} \pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2),$$  \hfill (15)

where $P(s_t = s|s_0, a_0; \pi)$ stands for the probability of reaching state $s$ at time step $t$, starting at $s_0$ and $a_0$ and then following $\pi$. Continue, we have:

$$|d^t_{\pi_1}(s) - d^t_{\pi_2}(s)|$$

$$= \sum_{s_0, a_0} \pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_1) - \sum_{s_0, a_0} \pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2)$$

$$\leq \sum_{s_0, a_0} \pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_1) - \sum_{s_0, a_0} \pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2)$$

$$+ \sum_{s_0, a_0} \pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2) - \sum_{s_0, a_0} \pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2)$$

$$\leq \sum_{s_0} d^t_{\pi_1}(s_1) (P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)) + E_{s_0 \sim \rho} \sum_{a_0} |\pi_1(a_0|s_0) - \pi_2(a_0|s_0)|P(s_t = s|s_0, a_0; \pi_2)$$  \hfill (16)

Add $\sum_s$ on both sides of the above equality, we get the following inequality:

$$\sum_s |d^t_{\pi_1}(s) - d^t_{\pi_2}(s)|$$

$$\leq E_{s_0 \sim d_{\pi_1}} \sum_s |P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)| + E_{s_0 \sim \rho} |\pi_1(\cdot|s_0) - \pi_2(\cdot|s_0)|_1$$  \hfill (17)
We can apply similar operations on $P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)$ as follows:

$$
\mathbb{E}_{s_1 \sim d_{\pi_1}^1} \sum_s |P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)| \\
= \mathbb{E}_{s_1 \sim d_{\pi_1}^1} \sum_s \left| \sum_a \pi_1(a|s_1) P(s_t = s|s_1, a; \pi_1) - \pi_2(a|s_1) P(s_t = s|s_1, a; \pi_2) \right| \\
\leq \mathbb{E}_{s_1 \sim d_{\pi_1}^1} \sum_s \left| P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2) \right| + \mathbb{E}_{s_1 \sim d_{\pi_1}^1} \left[ \| \pi_1(\cdot|s_1) - \pi_2(\cdot|s_1) \|_1 \right]
$$

Again, if we continue expand $P(s_t = s|s_2; \pi_1) - P(s_t = s|s_2; \pi_2)$ till time step $t$, we get:

$$
\sum_s |d^t_{\pi_1}(s) - d^t_{\pi_2}(s)| \leq \sum_{i=0}^{t-1} \mathbb{E}_{s_i \sim d_{\pi_1}^1} \left[ \| \pi_1(\cdot|s_i) - \pi_2(\cdot|s_i) \|_1 \right]
$$

Hence, for $\| d_{\pi_1} - d_{\pi_2} \|_1$, we have:

$$
\| d_{\pi_1} - d_{\pi_2} \|_1 \leq (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \| d^t_{\pi_1} - d^t_{\pi_2} \|_1 \\
\leq \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s \sim d_{\pi_1}^1} \left[ \| \pi_1(\cdot|s) - \pi_2(\cdot|s) \|_1 \right] \leq \sum_{t=0}^{\infty} 2\gamma^t \mathbb{E}_{s \sim d_{\pi_1}^1} \left[ D_{TV}(\pi_1(\cdot|s), \pi_2(\cdot|s)) \right] \leq \frac{2\gamma}{1-\gamma}.
$$

**Lemma A.2** For any two distribution $P$ and $Q$ over $\mathcal{X}$, and any bounded function $f : \mathcal{X} \rightarrow \mathbb{R}$ such that $|f(x)| \leq c, \forall x \in \mathcal{X}$, we have:

$$
| \mathbb{E}_{x \sim P} [f(x)] - \mathbb{E}_{x \sim Q} [f(x)] | \leq c \| P - Q \|_1.
$$

**Proof:**

$$
| \mathbb{E}_{x \sim P} [f(x)] - \mathbb{E}_{x \sim Q} [f(x)] | = | \sum_{x \in \mathcal{X}} P(x)f(x) - Q(x)f(x) | \\
\leq \sum_{x \in \mathcal{X}} |P(x)f(x) - Q(x)f(x)| \leq \sum_{x} |f(x)||P(x) - Q(x)| \\
\leq c \sum_{x} |P(x) - Q(x)| = c \| P - Q \|_1.
$$

**A.2 Proof of Theorem 3.1**

Recall that we denote $d_{\pi}$ as the joint state-action distribution under policy $\pi$. To prove Theorem 3.1, we will use Lemma A.2 presented in the Appendix from [23] to prove the following claim:

**Lemma A.3** Suppose we learned a approximate model $\hat{P}$ and obtain the optimal policy $\eta_n$ with respect to the objective function $J(\eta)$ under $P$ and the trust-region constraint $\mathbb{E}_{s \sim d_{\pi_n}} D_{TV}(\pi, \pi_n) \leq \alpha$, then compare to $\pi^*_n$, we have:

$$
J(\eta_n) - J(\eta^*_n) \leq \frac{\gamma}{2(1-\gamma)} \left( \mathbb{E}_{(s,a) \sim d_{\pi_n} \eta_n} \left[ \| \hat{P}_{s,a} - P_{s,a} \|_1 \right] + \mathbb{E}_{(s,a) \sim d_{\pi_n} \eta_n} \left[ \| \hat{P}_{s,a} - P_{s,a} \|_1 \right] \right).
$$

**Proof:** Denote $\hat{V}^\pi$ as the value function of policy $\pi$ under the approximate model $\hat{P}$. From Lemma 1.2 and Corollary 1.2 in [23], we know that for any two policies $\pi_1$ and $\pi_2$, we have:

$$
J(\pi_1) - J(\pi_2) = \mathbb{E}_{s \sim d_\pi} [\hat{V}^\pi_1(s) - \hat{V}^\pi_2(s)] \\
+ \frac{\gamma}{2(1-\gamma)} \left( \mathbb{E}_{(s,a) \sim d_{\pi_1} \pi_1} \left[ \| \hat{P}_{s,a} - P_{s,a} \|_1 \right] + \mathbb{E}_{(s,a) \sim d_{\pi_2} \pi_2} \left[ \| \hat{P}_{s,a} - P_{s,a} \|_1 \right] \right).
$$
Now replace $\pi_1$ with $\eta_n$ and $\pi_2$ with $\eta_n^*$. Note that both $\eta_n$ and $\eta_n^*$ are in the trust region constraint $E_{s \sim d_{\pi_n}} D_{TV}(\pi, \pi_n) \leq \alpha$ by definition. As $\eta_n$ is the optimal control under the approximate model $\hat{P}$ (i.e., the optimal solution to Eq. 4), we must have $E_{s \sim \eta_n} [V_{\eta_n}(s) - \hat{V}_{\eta_n}(s)] \leq 0$. Substitute it back to Eq. 23, we immediately prove the above lemma.

The above lemma shows that the performance gap between $\eta_n$ and $\eta_n^*$ is measured under the state-action distributions measured from $\eta_n$ and $\eta_n^*$ while our model $\hat{P}$ is only accurate under the state-action distribution from $\pi_n$. Luckily due to the trust-region constraint $E_{s \sim d_{\pi_n}} D_{TV}(\pi, \pi_n)$ and the fact that $\eta_n$ and $\eta_n^*$ are both in the trust-region, we can show that $d_{\eta_n, \pi_n}, d_{\pi_n, \eta_n}^*$ are not that far from $d_{\pi_n, \pi_n}$ using Lemma A.1:

$$
\|d_{\eta_n, \pi_n} - d_{\pi_n, \pi_n}\|_1 \leq \|d_{\eta_n, \pi_n} - d_{\pi_n, \eta_n}\|_1 + \|d_{\pi_n, \pi_n} - d_{\pi_n, \pi_n}\|_1
\leq \|d_{\eta_n, \pi_n} - d_{\pi_n, \pi_n}\|_1 + E_{s \sim d_{\pi_n}} \|\eta_n(\cdot|s) - \pi_n(\cdot|s)\|_1 \leq \frac{2\alpha}{1-\gamma} + 2\alpha \leq \frac{4\alpha}{1-\gamma}.
$$

Similarly, for $\pi_n^*$ we have:

$$
\|d_{\eta_n^*, \pi_n^*} - d_{\pi_n, \pi_n}\|_1 \leq \frac{4\alpha}{1-\gamma}.
$$

Go back to Eq. 22, let us replace $E_{d_{\pi_n} \eta_n}$ and $E_{d_{\pi_n} \eta_n^*}$ by $E_{d_{\pi_n} \pi_n}$ and using Lemma A.2, we will have:

$$
E_{(s,a) \sim d_{\eta_n, \pi_n}} [\|\hat{P}_{s,a} - P_{s,a}\|_1] - E_{(s,a) \sim d_{\pi_n, \pi_n}} [\|\hat{P}_{s,a} - P_{s,a}\|_1] \leq \frac{8\alpha}{1-\gamma},
$$

and similarly,

$$
E_{(s,a) \sim d_{\eta_n^*, \pi_n^*}} [\|\hat{P}_{s,a} - P_{s,a}\|_1] \leq E_{(s,a) \sim d_{\pi_n, \pi_n}} [\|\hat{P}_{s,a} - P_{s,a}\|_1] + \frac{8\alpha}{1-\gamma}.
$$

Combine Eqs. 26 and 27, we have:

$$
J(\eta_n) - J(\eta_n^*) \leq \frac{\gamma}{2(1-\gamma)} \left( 2E_{(s,a) \sim d_{\pi_n, \pi_n}} [\|\hat{P}_{s,a} - P_{s,a}\|_1] + 16\alpha/(1-\gamma) \right)
= \frac{\gamma \delta}{1-\gamma} + \frac{8\gamma \alpha}{(1-\gamma)^2} = O \left( \frac{\gamma \delta}{1-\gamma} \right) + O \left( \frac{\gamma \alpha}{(1-\gamma)^2} \right).
$$

Using the definition of $\Delta(\alpha)$, adding $J(\pi_n)$ and subtracting $J(\pi_n)$ on the LHS of the above inequality, we prove the theorem.

A.3 Proof of Theorem 4.1

The definition of $\pi_{n+1}$ implies that $E_{s \sim d_{\pi_{n+1}}} D_{TV}(\pi_{n+1}(\cdot|s), \pi_n(\cdot|s)) \leq \beta$. Using Lemma A.1, we will have that the total variation distance between $d_{\pi_{n+1}}$ and $d_{\pi_n}$ is:

$$
\|d_{\pi_{n+1}} - d_{\pi_n}\|_1 \leq \frac{2\beta}{1-\gamma}.
$$

Now we can compute the performance improvement of $\pi_{n+1}$ over $\eta_n$ as follows:

$$
(1-\gamma)(J(\pi_{n+1}) - J(\eta_n)) = E_{s \sim d_{\eta_n}} E_{a \sim A_{\eta_n}(s,a)} [\alpha_{\eta_n}(s,a)]
= E_{s \sim d_{\eta_n}} E_{a \sim A_{\eta_n}(s,a)} [A_{\eta_n}(s,a)] - E_{s \sim d_{\eta_n}} E_{a \sim A_{\pi_{n+1}}(s,a)} [A_{\pi_{n+1}}(s,a)] + E_{s \sim d_{\eta_n}} E_{a \sim A_{\pi_{n+1}}(s,a)} [A_{\pi_{n+1}}(s,a)]
\leq E_{s \sim d_{\eta_n}} E_{a \sim A_{\eta_n}(s,a)} [A_{\eta_n}(s,a)] - E_{s \sim d_{\eta_n}} E_{a \sim A_{\pi_{n+1}}(s,a)} [A_{\pi_{n+1}}(s,a)] + E_{s \sim d_{\eta_n}} E_{a \sim A_{\pi_{n+1}}(s,a)} [A_{\pi_{n+1}}(s,a)]
\leq \frac{2\varepsilon \beta}{1-\gamma} + E_{s \sim d_{\eta_n}} E_{a \sim A_{\pi_{n+1}}(s,a)} [A_{\pi_{n+1}}(s,a)]
= \frac{2\varepsilon \beta}{1-\gamma} + \varepsilon_n(\pi_{n+1})
= \frac{2\varepsilon \beta}{1-\gamma} - \varepsilon_n(\pi_{n+1})
$$

(30)
Finally, to bound \( J(\pi_{n+1}) - J(\pi_n) \), we can simply do:
\[
J(\pi_{n+1}) - J(\pi_n) = J(\pi_{n+1}) - J(\eta_n) + J(\eta_n) - J(\pi_n) \\
\leq \frac{\beta \varepsilon}{(1 - \gamma)^2} - \frac{|\Delta_n(\pi_{n+1})|}{1 - \gamma} - \Delta(\alpha).
\]

(A.4) Global Performance Guarantee for DPI

When \(|\Delta_n(\alpha)|\) and \(|\Delta_n(\pi_{n+1})|\) are small, say \(|\Delta_n(\alpha)| \leq \varepsilon, |\Delta_n(\pi_{n+1})| \leq \varepsilon\), then we can guarantee that \(\eta_n\) and \(\pi_n\) are good policies, if our initial distribution \(\rho\) happens to be a good distribution (e.g., close to \(d_\pi\)), and the realizable assumption holds: \(\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi(s)} A^\rho(s,a)] = \mathbb{E}_{s \sim d_\pi} [\min_{a \sim A} A^\rho(s,a)]\). We call a policy class \(\Pi\) closed under its convex hull if for any sequence of policies \(\{\pi_i\}, \pi_i \in \Pi\), 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 A.4 Under the realizable assumption and the assumption of \(\Pi\) 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:
\[
J(\eta_n) - J(\pi^*) \leq \left( \max_s \left( \frac{d_{\pi_n}(s)}{\rho_0(s)} \right) \right) \left( \frac{\xi}{\beta(1 - \gamma)^2} + \frac{\xi}{\beta(1 - \gamma)} \right).
\]

Proof: Recall the average advantage of \(\pi_{n+1}\) over \(\pi_n\) is defined as \(\Delta_{\pi_n}(\pi_{n+1}) = \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi_{n+1}(s)} A^{\rho}(s,a)]\). Also recall that the conservative update where we first compute \(\pi_n = \arg \min_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi} A^\rho(s,a)]\), and then compute the new policy \(\pi'_{n+1} = (1 - \delta)\pi_n + \beta\pi^*\). Note that under the assumption that the policy class \(\Pi\) is closed under its convex hull, we have that \(\pi'_{n+1} \in \Pi\). As we showed that \(\pi'_{n+1}\) satisfies the trust-region constraint defined in Eq. 5, we must have:
\[
\Delta_{\pi_n}(\pi_{n+1}) = \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi_{n+1}(s)} A^{\rho}(s,a)] \leq \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi^*(s)} A^{\rho}(s,a)], \tag{32}
\]

due to the fact that \(\pi_{n+1}\) is the optimal solution of the optimization problem shown in Eq. 5 subject to the trust region constraint. Hence if \(\Delta_{\pi}(\pi_{n+1}) \geq -\xi\), we must have \(\mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi_{n+1}} A^{\rho}(s,a)] \geq -\xi\), which means that:
\[
\mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi_{n+1}} A^{\rho}(s,a)] = (1 - \beta)(1 - \gamma)J(\pi_n) + \beta \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi^*} A^{\rho}(s,a)] \geq -\xi.
\]
\[
\Rightarrow \mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi^*} A^{\rho}(s,a)] \geq -\xi - \frac{1 - \beta}{\beta}(1 - \gamma)\Delta(\alpha) \geq -\xi - \frac{1 - \gamma}{\beta} \Delta(\alpha). \tag{33}
\]

Recall the realizable assumption: \(\mathbb{E}_{s \sim d_\pi} [\mathbb{E}_{a \sim \pi^*} A^{\rho}(s,a)] = \mathbb{E}_{s \sim d_\pi} [\min_a A^{\rho}(s,a)]\), we have:
\[
\frac{-\xi}{\beta} - \frac{1 - \gamma}{\beta} \Delta(\alpha) \leq \sum_s d_{\pi_n}(s) \min_a A^{\rho}(s,a) = \sum_s \frac{d_{\pi_n}(s)}{d_{\pi^*}(s)} d_{\pi^*}(s) \min_a A^{\rho}(s,a) \\
\leq \min_s \left( \frac{d_{\pi_n}(s)}{d_{\pi^*}(s)} \right) \sum_s d_{\pi^*}(s) \min_a A^{\rho}(s,a) \\
\leq \min_s \left( \frac{d_{\pi_n}(s)}{d_{\pi^*}(s)} \right) \sum_s d_{\pi^*}(s) \sum_a \pi^*(a|s) A^{\rho}(s,a) \\
= \min_s \left( \frac{d_{\pi_n}(s)}{d_{\pi^*}(s)} \right) (1 - \gamma)(J(\pi^*) - J(\eta_n)). \tag{34}
\]
\[\]
Applying PDL, we have:

\[ J(\eta_n) - J(\pi^*) \leq \left( \max_s \left( \frac{d_{x^n}(s)}{d_{\pi_n}(s)} \right) \right) \left( \frac{\xi}{\beta(1-\gamma)} + \frac{\Delta(\alpha)}{\beta} \right) \]

\[ \leq \left( \max_s \left( \frac{d_{x^n}(s)}{\rho(s)} \right) \right) \left( \frac{\xi}{\beta(1-\gamma)^2} + \frac{\xi}{\beta(1-\gamma)} \right) \]  

(35)

\[ \square \]

A.5 Analysis on Using DAGGER for Updating \( \pi_n \)

Note that in ExIt, once an intermediate expert is constructed, DAGGER [25] is used as the imitation learning algorithm to improve the reactive policy. DAGGER does not directly optimize the ultimate objective function—the expected total cost, but instead trying minimizes the number of mismatches between the learner and the expert. Here we used more advanced, cost-aware IL algorithms, AGGREVA [24] and AGGREVAED [33], which directly reason about the expected total cost, and guarantee to learn a policy that achieves one-step deviation improvement of the expert policy.

Below we analyze the update of \( \pi \) using DAGGER.

To analyze the update of \( \pi \) using DAGGER, we consider deterministic policy here: we assume \( \pi_n \) and \( \eta \) are both deterministic and the action space \( \mathcal{A} \) is discrete. We consider the following update procedure for \( \pi \):

\[ \min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{x^n}} \left[ \mathbb{E}_{a \sim \pi - \eta + \pi_n(s, a)} \right], \]

s.t., \( \mathbb{E}_{s \sim d_{x^n}} \left[ \| \pi(\cdot) - \pi_n(\cdot) \| \right] \leq \beta. \]  

(36)

Namely we simply convert the cost vector defined by the disadvantage function by a “one-hot” encoded cost vector, where all entries are 1, except the entry corresponding to \( \arg \min_a A^{\eta_n}(s, a) \) has cost 0. Ignoring the updates on the “expert” \( \eta_n \), running the above update step with respect to \( \pi \) can be regarded as running online gradient descent with a local metric defined by the trust-region constraint. Recall that \( \eta_n \) may from a different policy class than \( \Pi \).

Assume that we learn a policy \( \pi_{n+1} \) that achieves \( \epsilon_n \) prediction error:

\[ \mathbb{E}_{s \sim d_{x^n}} \left[ \mathbb{E}_{a \sim \pi - \eta + \pi_n(s, a)} \right] \leq \epsilon_n. \]  

(37)

Namely we assume that we learn a policy \( \pi_{n+1} \) such that the average probability of mismatch to \( \eta_n \) is at most \( \epsilon_n \).

Using Lemma A.1, we will have that the total variation distance between \( d_{\pi_{n+1}} \) and \( d_{\pi_n} \) is at most:

\[ \| d_{\pi_{n+1}} - d_{\pi_n} \| \leq \frac{2\beta}{1 - \gamma}. \]  

(38)

Applying PDL, we have:

\[ (1 - \gamma) \left( J(\pi_{n+1}) - J(\eta_n) \right) = \mathbb{E}_{s \sim d_{x^n+1}} \left[ \mathbb{E}_{a \sim \pi_{n+1}} \left[ A^{\eta_n}(s, a) \right] \right] \]

\[ \leq \mathbb{E}_{s \sim d_{x^n}} \left[ \max_{a \neq \arg \min_a A^{\eta_n}(s, a)} \pi(a | s) A^{\eta_n}(s, a) \right] + \frac{2\beta\epsilon}{1 - \gamma} \]

\[ = \mathbb{E}_{s \sim d_{x^n}} \left[ \max_{a \neq \arg \min_a A^{\eta_n}(s, a)} \pi(a | s) A^{\eta_n}(s, a) \right] + \frac{2\beta\epsilon}{1 - \gamma} \]

\[ \leq \epsilon' \epsilon_n + \frac{2\beta\epsilon}{1 - \gamma}, \]  

(39)

where we define \( \epsilon' = \max_{s, a} A^{\eta_n}(s, a) \), which should be at a similar scale as \( \epsilon \). Hence we can show that performance difference between \( \pi_{n+1} \) and \( \pi_n \) as:

\[ J(\pi_{n+1}) - J(\pi_n) \leq \frac{2\beta\epsilon}{(1 - \gamma)^2 + \frac{\epsilon' \epsilon_n}{1 - \gamma} - \Delta(\alpha)}. \]  

(40)
Now we can compare the above upper bound to the upper bound shown in Theorem 4.1. Note that even if we assume the policy class is rich and the learning process perfect learns a policy (i.e., \( \pi_{n+1} = \eta_n \)) that achieves prediction error \( \epsilon_n = 0 \), we can see that the improvement of \( \pi_{n+1} \) over \( \pi_n \) only consists of the improvement from the local optimal control \( \Delta(\alpha) \). While in theorem 4.1, under the same assumption, except for \( \Delta(\alpha) \), the improvement of \( \pi_{n+1} \) over \( \pi_n \) has an extra term \( \frac{1}{1+\gamma} |A_n(\pi_{n+1})| \), which basically indicates that we learn a policy \( \pi_{n+1} \) that is one-step deviation improved over \( \eta_n \) by leveraging the cost informed by the disadvantage function. If one uses DAgger, than the best we can hope is to learn a policy that performs as good as the “expert” \( \eta_n \) (i.e., \( \epsilon_n = 0 \)).

### B Additional Experimental Details

#### B.1 Synthetic Discrete MDPs and Conservative Policy Update Implementation

We follow [27] to randomly create 10 discrete MDPs, each with 1000 states, 5 actions and 2 branches (namely, each state-action pair leads to at most 2 different states in the next step). We work in model-free setting: we cannot explicitly compute the distribution \( d_\pi \) and we can only generate samples from \( d_\pi \) by executing \( \pi \).

We maintain a tabular representation \( \hat{P} \in \mathbb{R}^{|S| \times |A| \times |S|} \), where each entry \( P_{i,j,k} \) records the number of visits of the state-action-next state triple. We represent \( \eta \) as a 2d matrix \( \eta \in \mathbb{R}^{|S| \times |A|} \), where \( \eta_{i,j} \) stands for the probability of executing action \( j \) at state \( i \). The reactive policy uses the binary encoding of the state id as the feature, which we denote as \( \phi(s) \in \mathbb{R}^{d_k} \) (\( d_k \) is the dimension of feature space, which is \( \log_2(|S|) \) in our setup). Hence the reactive policy \( \pi_n \) sits in low-dimensional feature space and doesn’t scale with respect to the size of the state space \( S \).

For both our approach and CPI, we implement the unconstrained cost-sensitive classification (Eq. 5) by the Cost-Sensitive One Against All (CSOAA) classification technique. Specifically, given a set of states \( \{s_i\} \), sampled from \( d_{\pi_n} \), and a cost vector \( \{A^{\pi_n}(s_i, \cdot) \in \mathbb{R}^{|A|}\} \) (byproduct of VI), we train a linear regressor \( \tilde{W} \in \mathbb{R}^{|A| \times d_k} \) to predict the cost vector: \( \tilde{W}\phi(s) \approx A^{\pi_n}(s, \cdot) \). Then \( \pi_n^* \) in Eq. 6 is just a classifier that predicts action \( \arg\min_i (\tilde{W}s)[i] \) corresponding to the smallest predicted cost. We then combine \( \pi_n^* \) with the previous policies as shown in Eq. 6 to make sure \( \pi_{n+1} \) satisfies the trust region constraint in Eq. 5.

For CPI, we estimate \( A^{\pi_n}(s, \alpha) \) by running value iteration using \( \hat{P} \) with the original cost matrix. We also experimented estimating \( A^{\pi_n}(s, \cdot) \) by empirical rollouts with importance weighting, which did not work well in practice due to high variance resulting from the empirical estimate and importance weight. For our method, we alternately compute \( \eta_n \) using VI with the new cost shown in Eq. 10 and \( \hat{P} \), and update the Lagrange multiplier \( \mu \), under convergence. Hence the only difference between our approach and CPI here is simply that we use \( A^{\pi_n} \) while CPI uses \( A^{\pi_n} \). We tuned the step size \( \beta \) (Eq. 6) for CPI. Neither our method nor CPI used line-search trick for \( \beta \).

Our results indicates that using \( A^{\pi_n} \) converges much faster than using \( A^{\pi_n} \), although computing \( \eta_n \) is much more time consuming than computing \( A^{\pi_n} \). But again we emphasize that computing \( \eta_n \) doesn’t require extra samples. For real large discrete MDPs, we can easily plug in an approximate VI techniques such as [12] to significantly speed up computing \( \eta_n \).

#### B.2 Details for Updating Lagrange Multiplier \( \mu \)

Though running gradient ascent on \( \mu \) is theoretically sound and can work in practice as well, but it converges slow and requires to tune the learning rate as we found experimentally. To speed up convergence, we used the same update procedure used in the practical implementation of Guided Policy Search [11]. We set up \( \mu_{\text{min}} \) and \( \mu_{\text{max}} \). Starting from \( \mu = \mu_{\text{min}} \), we fix \( \mu \) and compute \( \eta \) using the new cost \( c' \) as shown in Eq. 10 under the local dynamics \( \hat{P} \) using LQR. We then compare \( E_{s \sim P_n} D_{KL}(\eta(\cdot|s), \pi_n(\cdot|s)) \) to \( \alpha \). If \( \eta \) violates the constraint, i.e., \( E_{s \sim P_n} D_{KL}(\eta(\cdot|s), \pi_n(\cdot|s)) > \alpha \), then it means that \( \mu \) is too small. In this case, we set \( \mu_{\text{min}} = \mu \), and compute new \( \mu \) as \( \mu = \min(\sqrt{\mu_{\text{min}} \mu_{\text{max}}}, 10 \mu_{\text{min}}) \). On the other hand, if \( \eta \) satisfies the KL constraint, i.e., \( \mu \) is too big, we set \( \mu_{\text{max}} = \mu \), and compute new \( \mu \) as \( \mu = \max(\sqrt{\mu_{\text{min}} \mu_{\text{max}}}, 0.1 \mu_{\text{max}}) \). We early terminate the process once we find \( \eta \) such that \( 0.9 \alpha \leq E_{s \sim P_n} D_{KL}(\eta(\cdot|s), \pi_n(\cdot|s)) \leq 1.1 \alpha \). We then store the most recent Lagrange multiplier \( \mu \) which will be used as warm start of \( \mu \) for the next iteration.
B.3 Details on the Natural Gradient Update to $\pi$

Here we provide details for updating $\pi_\theta$ via imitating $\eta_\theta$ (Sec. 5.2). Recall the objective function $L_n(\theta) = E_{s\sim d_{\pi_n}} [E_{a\sim \pi(\cdot|s,\theta)} [A^\eta_{\pi_n} (s, a)]]$, $\nabla_{\theta_n} L_n(\theta) = \nabla_{\theta_n} L_n(\theta)|_{\theta=\theta_n}$, and $F_{\theta_n}$ is the Fisher information matrix (equal to the Hessian of the KL constraint measured at $\theta_n$). To compute $F_{\theta_n}^{-1} \nabla_{\theta_n}$, in our implementation, we use Conjugate Gradient with the Hessian-vector product trick [28] to directly compute $F_{\theta_n}^{-1} \nabla_{\theta_n}$.

Note that the unbiased empirical estimation of $\nabla_{\theta_n}$ and $F_{\theta_n}$ is well-studied and can be computed using samples generated from executing $\pi_{\theta_n}$. Assume we roll out $\pi_{\theta_n}$ to generate $K$ trajectories $\tau^i = \{s^i_0, a^i_0, \ldots, s^i_T, a^i_T\}, \forall i \in [K]$. The empirical gradient and Fisher matrix can be formed using these samples as $\nabla_{\theta_n} = \sum_{s,a} [\nabla_{\theta_n} (\ln(\pi(a|s;\theta_n))) A^\eta_{\pi_n} (s, a)]$ and $F_{\theta_n} = \sum_{s,a} [\nabla (\ln(\pi(a|s;\theta_n))) (\nabla_{\theta_n} \ln(\pi(a|s;\theta_n)))^T]$.

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. 12, with the learning rate shrinking to $\sqrt{(\beta/k)/(\nabla_{\theta_n}^T F_{\theta_n}^{-1} \nabla_{\theta_n})}$ to ensure that after $k$ steps, the solution still satisfies the constraint in Eq. 11.

B.4 Details on the Continuous Control Experiment Setup

The cost function $c(s, a)$ for discrete MDP is uniformly sampled from $[0, 1]$. For the continuous control experiments, we designed the cost function $c(s, a)$, which is set to be known to our algorithms. For cartpole and helicopter hover, denote the target state as $s^*$, the cost function is designed to be exactly quadratic: $c(s, a) = (s - s^*)^T Q (s - s^*) + a^T Ra$, which penalizes the distance to the goal and large control inputs. For Swimmer, Hopper and Half-Cheetah experiment, we set up a target moving forward speed $v^*$. For any state, denote the velocity component as $s_v$, the quadratic cost function is designed as $c(s, a) = q(s_v - v^*)^2 + a^T Ra$, which encourages the agent to move forward in a constant speed while avoiding using large control inputs.

For reactive policies, we simply used two-layer feedforward neural network as the parameterized policies—the same structures used in the implementation of [28].

For local linear model fit under $d_{\pi}$, given a dataset in the format of $\{s_i, a_i, s'_i\}_{i=1}^N$, where $s_i \sim d_{\pi}$, $a_i \sim \pi(\cdot|s)$, and $s'_i \sim P_{s_i, a_i}$, we perform Ridge linear regression:

$$A, B, c = \arg \min_{A, B, c} \frac{1}{N} \sum_{i=1}^N ||A s_i + B a_i + c - s'_i||^2_2 + \lambda (||A||_F + ||B||_F + ||c||_2),$$

(41)

where regularization $\lambda$ is pre-fixed to be a small number for all experiments. With $A, B, c$, we estimate $\Sigma$ as $\Sigma = \frac{1}{N} \sum_{i=1}^N e_i e_i^T$, where $e_i = A s_i + B a_i + c - s'_i$. 

18