Dual Policy Iteration

Wen Sun [1] **Geoffrey J. Gordon** [2] **Byron Boots** [3] **J. Andrew Bagnell** [1]

Abstract

Recently, a novel class of Approximate Policy Iteration (API) algorithms have demonstrated impressive practical performance (e.g., ExIt from Anthony et al. (2017)). 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, but is only available during training. The reactive policy is updated under supervision from the non-reactive policy, while the non-reactive policy is improved with guidance from the reactive policy. In this work we study this 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 for applications with unknown dynamics. We demonstrate the efficacy of our approach on various continuous control Markov Decision Processes (MDPs).

1. Introduction

Approximate Policy Iteration (API) (Bagnell et al., 2004; Kakade & Langford, 2002; Lazaric et al., 2010; Scherrer, 2014; Bertsekas & Tsitsiklis, 1995), including conservative API (Kakade & Langford, 2002), API driven by learned critics (Rummery & Niranjan, 1994), or gradient-based API with stochastic policies (Baxter & Bartlett, 2001; Bagnell & Schneider, 2003; Kakade, 2002; Schulman et al., 2015a), 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 (Bagnell et al., 2004; Kakade & Langford, 2002) 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 (Kakade & Langford, 2002), modifying only single time steps (Bagnell et al., 2004), or making small changes to policy parameters (Kakade, 2002; Bagnell & Schneider, 2003). Recently, a new class of API algorithms, which we call *Dual Policy Iteration* (DPI), 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. Ross & Bagnell (2014); Sun et al. (2017) 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 (Anthony et al., 2017) maintains and updates a UCT-based policy (Upper Confidence Bound applied to Tree (Kocsis & Szepesvári, 2006)) 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 superhuman performance at the ancient game of Go. Zucker & 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 It-

[1]Robotics Institute, Carnegie Mellon University, USA [2]Machine Learning Department, Carnegie Mellon University, USA [3]College of Computing, Georgia Institute of Technology, USA. Correspondence to: Wen Sun <wensun@cs.cmu.edu>.

eration and Differential Dynamic Programming (DDP) to construct the "expert" policy that improves upon the current reactive one. (Levine & Abbeel, 2014).

In this work, we provide a general framework for synthesizing and analyzing such Dual Policy Iteration 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 (Bagnell & Schneider, 2001; Atkeson, 2012) 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 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 (An et al., 1988; Abbeel et al., 2006; Kolter & Ng, 2009).

2. Preliminaries

Formally, 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 $P(\cdot|s,a)$. We denote $c(s,a)$ as the cost of taking action a while in state s. Finally, ρ_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 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 (Jaksch et al., 2010; Azar et al., 2017).

We define a stochastic policy π such that for any state $s \in \mathcal{S}$, $\pi(\cdot|s) \in \Delta(A)$, 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 π until and including t, is defined $\forall s_t$: $d_\pi^t(s_t) = \sum_{\{s_i,a_i\}_{i\le t-1}} \rho_0(s_0) \prod_{i=0}^{t-1} \pi(a_i|s_i) P(s_{i+1}|s_i,a_i)$, where by definition $d_\pi^0(s) = \rho_0(s)$ for any π. 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 $d_\pi \pi(s,a) = d_\pi(s)\pi(a|s)$, then the expected total discounted sum of costs of a policy π is: $J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s,a)\sim d_\pi^t \pi}[\gamma^t c(s,a)]$. We define the value function $V^\pi(s)$ and state-action value function $Q^\pi(s,a)$ as

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

where the expectation in the first equation is taken with respect to the randomness from the policy π and the MDP, and $P_{s,a}$ is short for $P(\cdot|s,a)$. With V^π and Q^π, 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^π as the *disadvantage* function.

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. Throughout the paper, we use the *Performance Difference* lemma (PDL) extensively:

Lemma 2.1 (Lemma 6.1(Kakade & Langford, 2002)). *For any two policies π and π', we have:*

$$J(\pi) - J(\pi') = \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim d_\pi \pi}\left[A^{\pi'}(s,a)\right]. \quad (1)$$

3. Dual-Policy Approximate Policy Iteration

In this section, we propose an alternating optimization framework for Dual-Policy API, inspired by the PDL (Lemma 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 min-max optimization framework:

$$\min_{\pi\in\Pi}\max_{\eta\in\Pi}\mathbb{E}_{s\sim d_\pi}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}\left[A^\eta(s,a)\right]\right]. \quad (2)$$

Note that the unique Nash equilibrium for the above equation is $(\pi,\eta) = (\pi^*,\pi^*)$. To show this, we can simply apply the PDL:

$$\begin{aligned}&\arg\min_{\pi\in\Pi}\arg\max_{\eta\in\Pi}\mathbb{E}_{s\sim d_\pi}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}\left[A^\eta(s,a)\right]\right]\\ =&\arg\min_{\pi\in\Pi}J(\pi) - \arg\min_{\eta\in\Pi}J(\eta). \quad (3)\end{aligned}$$

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), π could stand for the fast reactive policy and η could correspond to the tree search policy. Below we first consider how we can update π given a fixed η. For notation purposes, we use π_n and η_n to represent the two policies in the n^{th} iteration.

3.1. Updating π

We update π_n to π_{n+1} by performing the following constrained optimization procedure:

$$\arg\min_{\pi}\mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}\left[A^{\eta_n}(s,a)\right]\right], \quad (4)$$

$$s.t., \mathbb{E}_{s\sim d_{\pi_n}}\left[D_{TV}(\pi(\cdot|s),\pi_n(\cdot|s))\right] \le \beta \quad (5)$$

Note that our formulation in Eq. 4 is different from existing API algorithms (e.g., CPI) which use the disadvantage function $A^{\pi_n}(s,a)$ of the current policy π_n while in our case, we use η_n's disadvantage function. As Eq. 4 is performing API with respect to a fixed policy η_n, we can solve Eq. 4 by converting it to supervised learning problem such as cost-sensitive classification (Kakade & Langford, 2002), subject to an L1 constraint. It is worth pointing out that the trust region definition in Eq. 5 is empirically measurable using samples from d_{π_n} (i.e., executing π_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 $\max_{s\in\mathcal{S}} D_{TV}(\pi(\cdot|s),\pi_n(\cdot|s))$, which is not empirically measurable unless one can enumerate all states.[1]

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. 4 without considering the constraint (Eq. 5) to obtain a policy π_n^*. We then conservatively update π_n to π_{n+1} as:

$$\pi_{n+1} = (1-\beta)\pi_n + \beta\pi_n^*. \quad (6)$$

Note that π_{n+1} satisfies the constraint as $D_{TV}(\pi_{n+1}(\cdot|s),\pi_n(\cdot|s)) \le \beta, \forall s$. Intuitively, a conservative update can be understood as first solving the objective function to obtain π_n^* without considering the constraint, and then move π_n towards π_n^* until the boundary of the region defined by the constraint is reached. For a parameterized policy, in Sec. 4.1, we will introduce a corresponding natural gradient update procedure.

3.2. Updating η

In this section, we present how to update η given a fixed π. Given π_n, the objective function for η in Eq. 2 becomes:

$$\max_{\eta}\mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_n(\cdot|s)}\left[A^\eta(s,a)\right]\right]. \quad (7)$$

While computing the functional gradient with respect to η measured at η_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 π' is equivalent to finding the optimal policy π^*: $\arg\max_\eta (J(\pi_n) - J(\eta)) \equiv \arg\min_\eta J(\eta)$, regardless of what π_n is. As directly minimizing the objective $J(\eta)$ is as hard as the original problem, we update η locally by constraining it to a trust region around π_n:

$$\arg\min_{\eta} J(\eta), \quad (8)$$

$$s.t., \mathbb{E}_{s\sim d_{\pi_n}} D_{TV}[(\eta(\cdot|s),\pi_n(\cdot|s))] \le \alpha. \quad (9)$$

3.2.1. Updating via Model-based RL

The constrained optimization problem in Eq. 8&9 is non-trivial, 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 π_n at the last optimization direction for π. Moreover, thanks to the trust region, we

[1] In fact, Schulman et al. (2015a) suggests relaxing $\max_{s\in\mathcal{S}}$ to $\mathbb{E}_{s\sim d_{\pi_n}}$ for practical implementations of TRPO.

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. 8 & Eq. 9) under the *real* model $P_{s,a}$ as η_n^*. Note that, due to the definition of the optimality, η_n^* must perform better than π_n: $J(\pi_n) - J(\eta_n^*) \geq \Delta_n(\alpha)$, where $\Delta_n(\alpha) \geq 0$ is the performance gain from η_n^* over π_n. Here the size of $\Delta_n(\alpha)$ depends on the size of the trust-region. When the trust region expands, i.e., α increases, then $\Delta_n(\alpha)$ approaches the performance difference between the optimal policy π^* and π_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* π_n: we seek a model $\hat{P}$, such that the quantity $\mathbb{E}_{(s,a)\sim d_{\pi_n}\pi_n} D_{TV}(\hat{P}_{s,a}, P_{s,a})$ 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:

$$\begin{aligned}&\arg\min_{\hat{P}\in\mathbf{P}} \mathbb{E}_{s\sim d_{\pi_n}, a\sim\pi_n(s)} D_{KL}(P_{s,a}, \hat{P}_{s,a}) \\ &= \arg\min_{\hat{P}\in\mathbf{P}} \mathbb{E}_{s\sim d_{\pi_n}, a\sim\pi_n(s), s'\sim P_{s,a}}[-\log \hat{P}_{s,a}(s')], \quad (10)\end{aligned}$$

where we denote $\mathbf{P}$ 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 π_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. 10 and assume $\hat{P}$ is δ-optimal:

$$\mathbb{E}_{(s,a)\sim d_{\pi_n}\pi_n} D_{TV}(\hat{P}_{s,a}, P_{s,a}) \leq \delta, \quad (11)$$

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

After achieving a locally accurate model $\hat{P}$, we can solve Eq. 8 using any existing stochastic MBOC solvers. Assume a MBOC solver returns an optimal policy η_n under the estimated model $\hat{P}$ and the trust-region constraint:

$$\begin{aligned}&\eta_n = \arg\min_{\pi} J(\pi), \\ &s.t.,\ s_{t+1} \sim \hat{P}_{s_t,a_t},\ \mathbb{E}_{s\sim d_{\pi_n}} D_{TV}(\pi, \pi_n) \leq \alpha. \quad (12)\end{aligned}$$

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

Theorem 3.1. *Assume $\hat{P}_{s,a}$ satisfies Eq. 11, and η_n is the output of a MBOC solver for the optimization problem defined in Eq. 12, then we have:*

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

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., δ is small (e.g., $\mathbf{P}$ is rich enough and we have enough data from d_{π_n}), α is small, and there exists a local optimal solution that is significantly better than the current policy π_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 η_n that significantly outperforms π_n.

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 $\mathbb{A}_n(\pi_{n+1})$ as the disadvantage of π^{n+1} over η_n under the state distribution d_{π_n}:

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

Note that $\mathbb{A}_n(\pi_{n+1})$ is at least as non-positive (if π and η are from the same function class, or π's policy class is rich enough to include η), as if we set π_{n+1} to η_n. In that case we simply have $\mathbb{A}_n(\pi_{n+1}) = 0$, which means we can hope that the trust-region optimization procedure (Eq. 4) finds a policy π_{n+1} that achieves $\mathbb{A}_n(\pi_{n+1}) < 0$. The question we want to answer is: by *how much* is the performance of π_{n+1} improved over π_n by solving the two trust-region optimization procedures detailed in Eq. 8 and Eq. 4. Following Theorem 4.1 from Kakade & Langford (2002), we define $\varepsilon = \max_s |\mathbb{E}_{a\sim\pi_{n+1}(\cdot|s)}[A^{\eta_n}(s,a)]|$, which intuitively measures the maximum possible one-step improvement one can achieve from η_n. The following theorem states the performance improvement from π_n to π_{n+1}:

Theorem 3.2. *Solve Eq. 8 to compute η_n and Eq. 4 to compute π_{n+1}. Then, the improvement of π_{n+1} over π_n is:*

$$\begin{aligned}&J(\pi_{n+1}) - J(\pi_n) \\ \leq&\frac{\beta\varepsilon}{(1-\gamma)^2} - \frac{|\mathbb{A}_n(\pi_{n+1})|}{1-\gamma} - \Delta_n(\alpha). \quad (13)\end{aligned}$$

The proof of Theorem 3.2 is provided in Appendix A.3. When β is small, we are guaranteed to find a policy π_{n+1} where the total cost decreases by $\Delta_n(\alpha) + |\mathbb{A}_n(\pi_{n+1})|/(1-\gamma)$ compared to π_n. Note that classic API performance improvement (Kakade & 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. 13. Hence Dual Policy API boosts the performance improvement by introducing an extra term $\Delta(\alpha)$.

Theorem 3.2 simply assumes that η_n is the optimal solution of Eq. 8 under real model $P_{s,a}$. When using MBOC with $\hat{P}$ to compute η_n that approximates the true optimal solution, using Theorem 3.1 together with Theorem 3.2, we can easily show that that π_{n+1} improves π_n by:

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

When π_n is far away from the optimal solution π^*, i.e., at the beginning of the learning process, one can expect $|\Delta_n(\alpha)|$ and $|\mathbb{A}_n(\pi_{n+1})|$ to have large magnitude. When $|\Delta_n(\alpha)|$ is small, say $|\Delta_n(\alpha)| \leq \xi$ for some small positive real number ξ, then it means that π_n is already an ϵ-locally optimal solution, where we define a policy π_n to be ϵ-locally optimal if and only if there exists a positive real number α 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 $|\mathbb{A}_n(\pi_{n+1})| \leq \xi$ also holds, then we can guarantee that η_n and π_n are good policies. Under the realizable assumption (i.e., Π is rich):

$$\begin{aligned}&\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}[A^{\eta_n}(s,a)]\right]\\ =&\mathbb{E}_{s\sim d_{\pi_n}}\left[\min_{a\sim\mathcal{A}}[A^{\eta_n}(s,a)]\right],\end{aligned} \tag{14}$$

using the techniques from (Kakade & Langford, 2002), we can relate the performance of η_n to the optimal policy π^*. We call a policy class Π *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 Π.

Theorem 3.3. *Assume Eq. 14 holds and Π is closed under its convex hull, and $\max\{|\mathbb{A}_n(\pi_{n+1})|, \Delta(\alpha)\} \leq \xi \in \mathbb{R}^+$, then for η_n, we have:*

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

The proof is provided in Appendix A.4. The term $(\max_s (d_{\pi^*}(s)/\rho_0(s)))$ measures the distribution mismatch between the initial state distribution ρ_0 and the optimal policy π^*, and appears in previous API algorithms such as CPI (Kakade & Langford, 2002) and PSDP (Bagnell et al., 2004). Although this shows that to guarantee good performance, one needs a ρ_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 $\mathbb{A}_n$ with Theorem 3.2, we show when either $|\Delta_n(\alpha)|$ or $|\mathbb{A}_n(\pi_{n+1})|$ have large magnitudes, π_{n+1} can improve over π_n significantly; when $|\Delta_n(\alpha)|$ and $|\mathbb{A}_n(\pi_{n+1})|$) are small, then π_n and η_n are already good policies.

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 η_n plays the role of the tree-based policy, and our π_n plays the role of the apprentice policy. We force η to stay close to π with a trust region (for the purpose of the tractability of dynamics learning and MBOC), while ExIt ensures it by forming η as the weighted mixing of the search tree and the apprentice policy. One major difference in updating π is that ExIt uses DAgger (Ross et al., 2011) to update π by attempting to minimize the counts of mismatches with respect to η, while we perform cost-sensitive classification with loss formed by the disadvantage vector $A^\eta(s,\cdot) \in \mathbb{R}^{|\mathcal{A}|}$, which enables us to link the imitation performance directly to the ultimate performance $J(\pi)$, and allows π to achieve a one-step deviation improvement over η_n (Ross & Bagnell, 2014; Sun et al., 2017). We provide a detailed analysis of using DAgger versus a cost-sensitive update with cost formed by A^η in Appendix A.5.

Our framework is also related to Imitation Learning (IL). IL usually assumes an optimal policy π^* is available to access *only* during training and the goal is to leverage π^* to quickly train a policy π that can perform well during test time when π^* is not available. When π^* is available during training, we can simply set $\eta_n = \pi^*$ (i.e., π^* is the solution of MBOC in Eq. 8&9 with $\alpha = \infty$) and thus reveals previous IL algorithm AggreVaTeD (Sun et al., 2017).

On the other hand, when solving MBOC in Eq. 8&9 with $\alpha = 0$, then we have $\eta_n = \pi_n$ and the update procedure for π_n in Eq. 4&5 reveals API (e.g., CPI or Natural Policy Gradient).

4. 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 $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ and action space $\mathcal{A} \subseteq \mathbb{R}^{d_a}$. We work on parameterized policies: we parameterize policy π as $\pi(\cdot|s;\theta)$ for any $s \in \mathcal{S}$ (e.g., a neural network with parameter θ), and parameterize η by a sequence of time-varying linear-Gaussian policies $\eta = \{\eta_t\}_{0\leq t\leq H}$, where $\eta_t(a|s) = \mathcal{N}(K_t s + k_t, P_t)$ with $K_t \in \mathbb{R}^{d_a\times d_s}$, $k_t \in \mathbb{R}^{d_a}$ and $P_t \in \mathbb{R}^{d_a\times d_a}$ as the parameters of η_t. 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 $D_{TV}(\pi, \pi_n)$ with the KL-divergence $D_{KL}(\pi_n, \pi)$, which allows us to leverage Natural Gradient (Kakade, 2002; Bagnell & Schneider, 2003) for updating θ_n.[2] To summarize, π_n and η_n are short for π_{θ_n} and $\eta_{\Theta_n} = \{\mathcal{N}(K_t s + k_t, P_t)\}_t$, respectively.

4.1. Updating π_θ

In policy space, the objective function shown in Eq. 4 is linear with respect to the policy to optimize π. However, the objective can easily become nonconvex with respect to parameters θ of the policy π. Performing a second order Taylor expansion of the KL constraint around θ_n, we get the following constrained optimization problem:

$$\min_\theta \mathbb{E}_{s\sim d_{\pi_{\theta_n}}}[\mathbb{E}_{a\sim\pi(\cdot|s;\theta)}[A^{\eta_{\Theta_n}}(s,a)]], \\ s.t., (\theta-\theta_n)^T F_{\theta_n}(\theta-\theta_n) \le \beta, \tag{15}$$

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

$$\theta_{n+1} = \theta_n - \mu F_{\theta_n}^{-1} \nabla_{\theta_n}, \tag{16}$$

where μ 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 θ, depending on the function approximator used for π_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. 16, 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. 15. In our implementation, we use Conjugate Gradient with the Hessian-vector product trick (Schulman et al., 2015a) to directly compute $F^{-1}\nabla$.

Note that the unbiased empirical estimation of ∇_{θ_n} and F_{θ_n} is well-studied in the literature and can be computed using samples generated from executing π_{θ_n}. Assume we roll out π_{θ_n} to generate K trajectories $\tau^i = \{s_0^i, a_0^i, ...s_T^i, a_T^i\}, \forall i \in [K]$. The empirical gradient and Fisher matrix can be formed using these samples as $\nabla_{\theta_n} = \sum_{s,a}\left[\nabla_{\theta_n}\left(\ln(\pi(a|s;\theta_n))\right) A^{\eta_{\Theta_n}}(s,a)\right]$ and $F_{\theta_n} = \sum_{s,a}\left[(\nabla \ln(\pi(a|s;\theta_n)))(\nabla_{\theta_n} \ln(\pi(a|s;\theta_n))^T\right]$.

4.2. Updating η_Θ

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

$$s_{t+1} \sim \mathcal{N}(A_t s_t + B_t a_t + c_t, \Sigma_t), \forall t \in [1, T], \tag{17}$$

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 π_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 $d_{\pi_n}\pi_n$).

Next, to find a locally optimal policy under the linear-Gaussian transitions (i.e., Eq. 12), we add the KL constraint to the objective with Lagrange multiplier μ and form an equivalent minmax problem:

$$\min_\eta \max_{\mu\ge 0} \mathbb{E}\left[\sum_{t=1}^T \gamma^{t-1} c(s_t, a_t)\right] \\ + \mu\Big(\sum_{t=1}^T \gamma^{t-1} \mathbb{E}_{s\sim d_\pi^t}[D_{KL}(\eta, \pi_n)] - \alpha\Big), \tag{18}$$

where μ is the Lagrange multiplier, which can be solved by alternatively updating π and μ (Levine & Abbeel, 2014). For a fixed μ, using the derivation from (Levine & Abbeel, 2014), ignoring terms that do not depend on π, the above formulation can be written as:

$$\arg\min_\eta \mathbb{E}\left[\sum_{t=1}^T \gamma^{t-1}(c(s_t,a_t)/\mu - \log \pi_n(a_t|s_t))\right] \\ - \sum_{t=1}^T \gamma^{t-1} \mathbb{E}_{s\sim d_\pi^t}[\mathcal{H}(\eta(\cdot|s))], \tag{19}$$

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:

$$c'(s_t, a_t) = c(s_t, a_t)/\mu - \log(\pi_n(a_t|s_t)), \tag{20}$$

Algorithm 1 AggreVaTeD-OC

1: **Input:** The given MDP
Parameters $\alpha \in \mathbb{R}^+$, $\beta \in \mathbb{R}^+$, $k \ge 1, k \in \mathbb{N}$
2: Initialize π_{θ_0}
3: **for** n = 0 to ... **do**
4: Execute π_{θ_n} to generate a set of trajectories
5: Fit dynamics $\hat{P}$ using $\{s_t, a_t, s_{t+1}\}$ (Eq. 17)
6: Solve the minmax Lagrangian in Eq. 18 subject to learned dynamics $\hat{P}$ and obtain η_{Θ_n}
7: Form disadvantage $A^{\eta_{\Theta_n}}$
8: Compute θ_{n+1} by k-steps of NGD (Eq. 16)
9: **end for**

[2] Small D_{KL} leads to small D_{TV} by Pinsker's inequality.

and an entropy regularization on π that encourages the diversity of the actions induced by π. 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. 20 can be found exactly by a Dynamic Programming (DP) based approach, the *Linear Quadratic Regulator* (LQR) (Kwakernaak & Sivan, 1972). Given a dataset $\{(s_t, a_t), c'(s_t, a_t)\}$ collected from executing π_n, we can learn a quadratic approximation of $c'(s, a)$ (Levine & Abbeel, 2014). With a quadratic approximation of c' and linear dynamics, we solve Eq. 20 for η exactly by LQR. Once we get η, we go back to Eq. 18 and update the Lagrange multiplier μ, for example, by projected gradient ascent (Zinkevich, 2003). Upon convergence, LQR gives us a sequence of controls in the format $\eta_{\Theta_n} = \{\mathcal{N}(K_t s_t + k_t, P_t); t \in [T]\}$, together with a sequence of quadratic cost-to-go functions $\{Q_t(s, a)\}_t$, where Q_t is in the format of $[s, a]^T F_t [s, a]/2 + q_t^T [s, a] + v_t$.[3] 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 η as an intermediate expert, the update step is similar to AggreVaTeD—a differential IL approach (Sun et al., 2017). Every iteration, we run π_{θ_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. 18 to compute η_{Θ_n} and $A^{\eta_{\Theta_n}}$ subject to the learned dynamics $\hat{P}$. We then perform NGD. We summarize the procedure, AggreVaTeD-OC, in Alg. 1.

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 (Bagnell & Schneider, 2001; Atkeson, 2012), 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 π_{θ_n} on environment i to generate a set of trajectories. For each environment i, following the MBOC approach introduced in Sec. 4.2, we learn a sequence of local linear Gaussian dynamics and compute a sequence of local linear Gaussian policies $\eta_{\Theta_n^i}$ and their associated disadvantages $A^{\eta_{\Theta_n^i}}, \forall i \in [M]$. With $A^{\eta_{\Theta_n^i}}, \forall i \in [M]$, following the NGD update introduced in Sec. 4.1, we consider all training environments equally and formalize the objective $L_n(\theta)$ as $L_n(\theta) = \sum_{i=1}^{M} \mathbb{E}_{s \sim d_{\pi_{\theta_n}}} [\mathbb{E}_{a \sim \pi(\cdot|s;\theta)} [A^{\eta_{\Theta_n^i}}]]$. We update θ_n to θ_{n+1} by computing the gradient $\nabla_\theta L_n(\theta)|_{\theta=\theta_n}$ and perform k-step NGD. We output a single policy π_{θ_n} at the end that can potentially be used for different test environments.

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

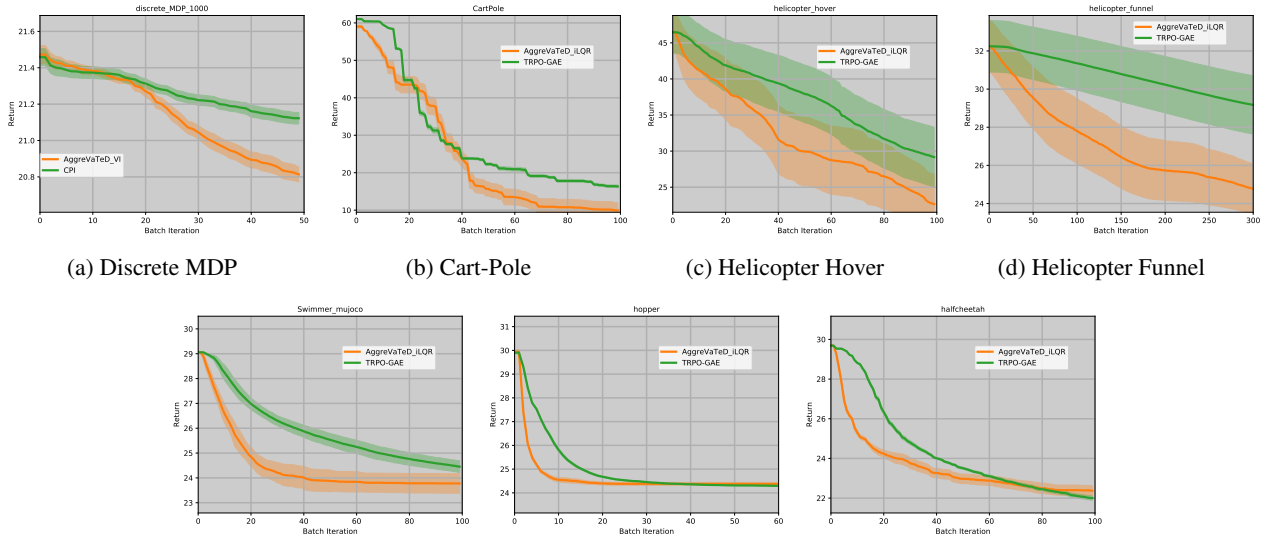
6. Experiments

We tested our approach on several MDPs: (1) a set of random discrete MDPs (Garnet problems (Scherrer, 2014)) (2) Cartpole balancing (Sutton & Barto, 1998), (3) Helicopter Aerobatics (Hover and Funnel) (Abbeel et al., 2005), (4) Swimmer, Hopper and Half-Cheetah from MuJoCo physics simulator (Todorov et al., 2012). The goals of these experiments are: (a) to experimentally verify that using A^η to perform API is more sample-efficient than using A^π. Although previous work, such as ExIt, has compared against REINFORCE (Williams, 1992) and experimentally provided an affirmative answer to this question, we would like to show the same phenomenon with η 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 (Atkeson, 2012). [4]

6.1. Comparison to CPI on Discrete MDP

Following (Scherrer, 2014), we randomly create ten discrete MDPs with 1000 states and 5 actions. Different from the techniques we introduced in Sec. 4.1 for continuous settings, here we use conservative update shown in Eq. 6, where each π_n^* is a linear classifier and is trained using regression-based cost-sensitive classification (Kakade & Langford, 2002). The feature for each state $\phi(s)$ is the binary encoding of the state ($\phi(s) \in \mathbb{R}^{\log_2(|\mathcal{S}|)}$). We maintain the estimated transition $\hat{P}$ in a tabular representation. The policy η 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 $|\mathcal{S}|$ and $|\mathcal{A}|$ 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. The setup and the conservative update implementation is detailed in Appendix B.1. Fig. 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^η instead of A^π for the policy update. The results indicates that performing policy iteration against a better policy speeds up the learning process.

[4] Link to our implementaion will be provide here.



(a) Discrete MDP (b) Cart-Pole (c) Helicopter Hover (d) Helicopter Funnel

(e) Swimmer (f) Hopper (g) Half-Cheetah

Figure 1. Performance (mean and standard error of cumulative cost in log-scale on y-axis) versus number of episodes (n on x-axis).

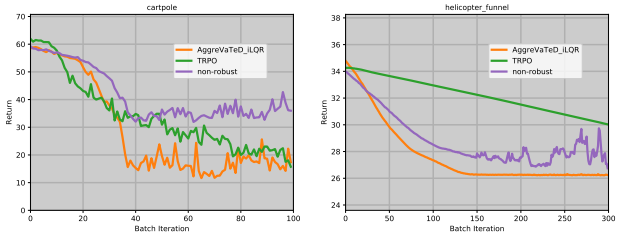
6.2. Comparison to Actor-Critic in Continuous Settings

We compare against TRPO-GAE on a set of continuous control tasks. The setup is detailed in Appendix B.3. 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 π which is updated by natural gradient descent. We use LQG as the underlying MBOC solver and we name our approach as AggreVaTeD-iLQG.

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 compared to TRPO-GAE. The major difference between our approach and TRPO-GAE is that we use A^η while TRPO-GAE uses A^π for the policy update. Note that both A^η and A^π are computed using the rollouts from π. The difference is that our approach uses rollouts to learn local dynamics and analytically estimates A^η using MBOC, while TRPO-GAE directly estimates A^π using rollouts. Overall, our approach converges faster than TRPO-GAE, which again indicates the benefit of using A^η for API.

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. 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 failures to further converge, which shows the overfitting phenomenon: the learned policy overfits to one particular training environment, which hurts the testing performance.



(a) Cart-Pole (b) Helicopter Funnel

Figure 2. Performance (mean in log-scale on y-axis) versus number of episodes (n on x-axis) in robust control.

7. Discussion and Future Work

We present and analyze a Dual Policy API strategy in an alternating optimization framework. We provide a simple instance of the framework which uses MBOC for updating the non-reactive policy and updates the reactive policy using natural gradient methods. Both our theoretical analysis

and emprical results suggest that our approach can outperform existing API algorithms.

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 (Venkatraman et al., 2015; Sun et al., 2016). 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.

References

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

Abbeel, Pieter, Quigley, Morgan, and Ng, Andrew Y. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 1–8. ACM, 2006.

An, Chae H, Atkeson, Christopher G, and Hollerbach, John M. *Model-based control of a robot manipulator*. MIT press, 1988.

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

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

Azar, Mohammad Gheshlaghi, Osband, Ian, and Munos, Rémi. Minimax regret bounds for reinforcement learning. *ICML*, 2017.

Bagnell, J Andrew and Schneider, Jeff. Covariant policy search. IJCAI, 2003.

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

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

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

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

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

Gorodetsky, Alex, Karaman, Sertac, and Marzouk, Youssef. 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.

Jaksch, Thomas, Ortner, Ronald, and Auer, Peter. Near-optimal regret bounds for reinforcement learning. *JMLR*, 2010.

Kakade, Sham. A natural policy gradient. *NIPS*, 2002.

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

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

Kolter, J Zico and Ng, Andrew Y. Policy search via the signed derivative. In *Robotics: science and systems*, pp. 34, 2009.

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

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

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

Ross, Stephane and Bagnell, Drew. Agnostic system identification for model-based reinforcement learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 1703–1710, 2012.

Ross, Stephane and Bagnell, J Andrew. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, J.Andrew. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Rummery, Gavin A and Niranjan, Mahesan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.

Scherrer, Bruno. Approximate policy iteration schemes: a comparison. In *International Conference on Machine Learning*, pp. 1314–1322, 2014.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael I, and Moritz, Philipp. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015a.

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

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

Sun, Wen, Venkatraman, Arun, Boots, Byron, and Bagnell, J Andrew. Learning to filter with predictive state inference machines. In *ICML*, 2016.

Sun, Wen, Venkatraman, Arun, Gordon, Geoffrey J, Boots, Byron, and Bagnell, J Andrew. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *ICML*, 2017.

Sutton, Richard S and Barto, Andrew G. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Venkatraman, Arun, Hebert, Martial, and Bagnell, J Andrew. Improving multi-step prediction of learned time series models. *AAAI*, 2015.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Zhou, Kemin, Doyle, John Comstock, Glover, Keith, et al. *Robust and optimal control*, volume 40. Prentice hall New Jersey, 1996.

Zinkevich, Martin. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *ICML*, 2003.

Zucker, Matt and Maas, Andrew. Learning tetris. 2009.

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:

$$\|P - Q\|_1 = 2D_{TV}(P, Q). \tag{21}$$

Recall that $d_\pi = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t d_\pi^t$. 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 π_1 and π_2 such that $\mathbb{E}_{s\sim d_{\pi_1}}\left[D_{TV}(\pi_1(\cdot|s), \pi_2(\cdot|s))\right] \leq \alpha$, then we have:*

$$\|d_{\pi_1} - d_{\pi_2}\|_1 \leq \frac{2\alpha}{1-\gamma}. \tag{22}$$

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

$$\begin{aligned}
&d_{\pi_1}^t(s) - d_{\pi_2}^t(s) \\
&= \sum_{s_0,s_1,..,s_{t-1}} \sum_{a_0,a_1,..,a_{t-1}} \Big(\rho(s_0)\pi_1(a_0|s_0)P_{s_0,a_0}(s_2)...\pi_1(a_{t-1}|s_{t-1})P_{s_{t-1},a_{t-1}}(s) \\
&- \rho(s_0)\pi_2(a_0|s_0)P_{s_0,a_0}(s_1)...\pi_2(a_{t-1}|s_{t-1})P_{s_{t-1},a_{t-1}}(s)\Big) \\
&= \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_1(a_0|s_0)\sum_{s_1}P_{s_0,a_0}(s_1)...\sum_{a_{t-1}}\pi_1(a_{t-1}|s_{t-1})P_{s_{t-1},a_{t-1}}(s) \\
&\quad - \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_2(a_0|s_0)\sum_{s_1}P_{s_0,a_0}(s_1)...\sum_{a_{t-1}}\pi_2(a_{t-1}|s_{t-1})P_{s_{t-1},a_{t-1}}(s) \\
&= \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_1) - \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2),
\end{aligned} \tag{23}$$

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 π. Continue, we have:

$$\begin{aligned}
&|d_{\pi_1}^t(s) - d_{\pi_2}^t(s)| \\
&= |\sum_{s_0}\rho(s_0)\sum_{a_0}\pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_1) - \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2)| \\
&\leq |\sum_{s_0}\rho(s_0)\sum_{a_0}\pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_1) - \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2)| \\
&\quad + |\sum_{s_0}\rho(s_0)\sum_{a_0}\pi_1(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2) - \sum_{s_0}\rho(s_0)\sum_{a_0}\pi_2(a_0|s_0)P(s_t = s|s_0, a_0; \pi_2)| \\
&\leq |\sum_{s_1} d_{\pi_1}^1(s_1)\left(P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)\right)| + \mathbb{E}_{s_0\sim\rho}\sum_{a_0}|\pi_1(a_0|s_0) - \pi_2(a_0|s_0)|P(s_t = s|s_0, a_0; \pi_2)
\end{aligned} \tag{24}$$

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

$$\begin{aligned}
&\sum_s |d_{\pi_1}^t(s) - d_{\pi_2}^t(s)| \\
&\leq \mathbb{E}_{s_1\sim d_{\pi_1}^1}\sum_s |P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)| + \mathbb{E}_{s_0\sim\rho}\|\pi_1(\cdot|s_0) - \pi_2(\cdot|s_0)\|_1
\end{aligned} \tag{25}$$

We can apply similar operations on $P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)$ as follows:

$$\begin{aligned}
&\mathbb{E}_{s_1 \sim d^1_{\pi_1}} \sum_s |P(s_t = s|s_1; \pi_1) - P(s_t = s|s_1; \pi_2)| \\
&= \mathbb{E}_{s \sim d^1_{\pi_1}} \sum_s |\sum_{a_1} [\pi_1(a_1|s_1) P(s_t = s|s_1, a_1; \pi_1) - \pi_2(a_1|s_1) P(s_t = s|s_1, a_2; \pi_2)]| \\
&\leq \mathbb{E}_{s_2 \sim d^2_{\pi_1}} \sum_s |P(s_t = s|s_2; \pi_1) - P(s_t = s|s_2; \pi_2)| + \mathbb{E}_{s_1 \sim d^1_{\pi_1}} [\|\pi_1(\cdot|s_1) - \pi_2(\cdot|s_1)\|_1]
\end{aligned}$$

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:

$$\sum_s |d^t_{\pi_1}(s) - d^t_{\pi_2}(s)| \leq \sum_{i=0}^{t-1} \mathbb{E}_{s_i \sim d^i_{\pi_1}} [\|\pi_1(\cdot|s_i) - \pi_2(\cdot|s_i)\|_1] \tag{26}$$

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

$$\begin{aligned}
&\|d_{\pi_1} - d_{\pi_2}\|_1 \leq (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \|d^t_{\pi_1} - d^t_{\pi_2}\|_1 \\
&\leq \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s \sim d^t_{\pi_1}} [\|\pi_1(\cdot|s) - \pi_2(\cdot|s)\|_1] \leq \sum_{t=0}^{\infty} 2\gamma^t \mathbb{E}_{s \sim d^t_{\pi_1}} [D_{TV}(\pi_1(\cdot|s), \pi_2(\cdot|s))] \leq \frac{2\alpha}{1-\gamma}.
\end{aligned} \tag{27}$$

□

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

$$|\mathbb{E}_{x \sim P}[f(x)] - \mathbb{E}_{x \sim Q}[f(x)]| \leq c\|P - Q\|_1. \tag{28}$$

Proof.

$$\begin{aligned}
&|\mathbb{E}_{x \sim P}[f(x)] - \mathbb{E}_{x \sim Q}[f(x)]| = |\sum_{x \in \mathcal{X}} P(x)f(x) - Q(x)f(x)| \\
&\leq \sum_x |P(x)f(x) - Q(x)f(x)| \leq \sum_x |f(x)||P(x) - Q(x)| \\
&\leq c \sum_x |P(x) - Q(x)| = c\|P - Q\|_1.
\end{aligned} \tag{29}$$

□

A.2. Proof of Theorem 3.1

Recall that we denote $d_\pi \pi$ as the joint state-action distribution under policy π. To prove Theorem 3.1, we will use Lemma 1.2 presented in the Appendix from Ross & Bagnell (2012) to prove the following claim:

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

$$J(\eta_n) - J(\pi^*_n) \leq \frac{\gamma}{2(1-\gamma)} \left(\mathbb{E}_{(s,a) \sim d_{\eta_n} \eta_n} \left[\|\hat{P}_{s,a} - P_{s,a}\|_1 \right] + \mathbb{E}_{(s,a) \sim d_{\pi^*_n} \pi^*_n} \left[\|\hat{P}_{s,a} - P_{s,a}\|_1 \right] \right). \tag{30}$$

Proof. Denote $\hat{V}^\pi$ as the value function of policy π under the approximate model $\hat{P}$. From Lemma 1.2 and Corollary 1.2 in Ross & Bagnell (2012), we know that for any two policies π_1 and π_2, we have:

$$\begin{aligned}
J(\pi_1) - J(\pi_2) &= \mathbb{E}_{s \sim \rho_0}[\hat{V}^{\pi_1}(s) - \hat{V}^{\pi_2}(s)] \\
&+ \frac{\gamma}{2(1-\gamma)} \left(\mathbb{E}_{(s,a) \sim d_{\pi_1} \pi_1} \left[\|\hat{P}_{s,a} - P_{s,a}\|_1 \right] + \mathbb{E}_{(s,a) \sim d_{\pi_2} \pi_2} \left[\|\hat{P}_{s,a} - P_{s,a}\|_1 \right] \right).
\end{aligned} \tag{31}$$

Now replace π_1 with η_n and π_2 with π_n^*. Note that both η_n and π_n^* are in the trust region constraint $\max_s D_{TV}(\pi, \pi_n) \leq \alpha$ by definition. As η_n is the optimal control under the approximate model $\hat{P}$ (i.e., the optimal solution to Eq. 12), we must have $\mathbb{E}_{s\sim\rho_0}[\hat{V}^{\pi_1}(s) - \hat{V}^{\pi_2}(s)] \leq 0$. Substitute it back to Eq. 31, we immediately prove the above lemma. □

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

$$\begin{aligned}
&\|d_{\eta_n}\eta_n - d_{\pi_n}\pi_n\|_1 \leq \|d_{\eta_n}\eta_n - d_{\pi_n}\eta_n\|_1 + \|d_{\pi_n}\eta_n - d_{\pi_n}\pi_n\|_1 \\
&\leq \|d_{\eta_n} - d_{\pi_n}\|_1 + \mathbb{E}_{s\sim d_{\pi_n}}[\|\eta_n(\cdot|s) - \pi_n(\cdot|s)\|_1] \leq \frac{2\alpha}{1-\gamma} + 2\alpha \leq \frac{4\alpha}{1-\gamma}.
\end{aligned} \tag{32}$$

similarly, for π_n^* we have:

$$\|d_{\pi_n^*}\pi_n^* - d_{\pi_n}\pi_n\|_1 \leq \frac{4\alpha}{1-\gamma}. \tag{33}$$

Go back to Eq. 30, let us replace $\mathbb{E}_{d_{\eta_n}\eta_n}$ and $\mathbb{E}_{d_{\pi_n^*}\pi_n^*}$ by $\mathbb{E}_{d_{\pi_n}\pi_n}$ and using Lemma A.2, we will have:

$$\begin{aligned}
&|\mathbb{E}_{(s,a)\sim d_{\eta_n}\eta_n}[\|\hat{P}_{s,a} - P_{s,a}\|_1] - \mathbb{E}_{(s,a)\sim d_{\pi_n}\pi_n}[\|\hat{P}_{s,a} - P_{s,a}\|_1]| \leq 2\|d_{\eta_n}\eta_n - d_{\pi_n}\pi_n\|_1 \leq \frac{8\alpha}{1-\gamma} \\
&\Rightarrow \mathbb{E}_{(s,a)\sim d_{\eta_n}\eta_n}[\|\hat{P}_{s,a} - P_{s,a}\|_1] \leq \mathbb{E}_{(s,a)\sim d_{\pi_n}\pi_n}[\|\hat{P}_{s,a} - P_{s,a}\|_1] + \frac{8\alpha}{(1-\gamma)},
\end{aligned} \tag{34}$$

and similarly,

$$\mathbb{E}_{(s,a)\sim d_{\pi_n^*}\pi_n^*}[\|\hat{P}_{s,a} - P_{s,a}\|_1] \leq \mathbb{E}_{(s,a)\sim d_{\pi_n}\pi_n}[\|\hat{P}_{s,a} - P_{s,a}\|_1] + \frac{8\alpha}{(1-\gamma)}. \tag{35}$$

Combine Eqs. 34 and 35, we have:

$$\begin{aligned}
J(\pi_n') - J(\pi_n^*) &\leq \frac{\gamma}{2(1-\gamma)}\left(2\mathbb{E}_{(s,a)\sim d_{\pi_n}\pi_n}[\|\hat{P}_{s,a} - P_{s,a}\|_1] + 16\alpha/(1-\gamma)\right) \\
&= \frac{\gamma\delta}{1-\gamma} + \frac{8\gamma\alpha}{(1-\gamma)^2} = O\left(\frac{\gamma\delta}{1-\gamma}\right) + O\left(\frac{\gamma\alpha}{(1-\gamma)^2}\right).
\end{aligned} \tag{36}$$

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 3.2

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

$$\|d_{\pi_{n+1}} - d_{\pi_n}\|_1 \leq \frac{2\beta}{1-\gamma}. \tag{37}$$

Now we can compute the performance improvement of π_{n+1} over η_n as follows:

$$\begin{aligned}
&(1-\gamma)(J(\pi_{n+1}) - J(\eta_n)) = \mathbb{E}_{s\sim d_{\pi_{n+1}}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] \\
&= \mathbb{E}_{s\sim d_{\pi_{n+1}}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] - \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] + \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] \\
&\leq \left|\mathbb{E}_{s\sim d_{\pi_{n+1}}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] - \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right]\right| + \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] \\
&\leq \frac{2\varepsilon\beta}{1-\gamma} + \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]\right] \\
&= \frac{2\varepsilon\beta}{1-\gamma} + \mathbb{A}_n(\pi_{n+1}) \\
&= \frac{2\varepsilon\beta}{1-\gamma} - |\mathbb{A}_n(\pi_{n+1})|
\end{aligned} \tag{38}$$

Finally, to bound $J(\pi_{n+1}) - J(\pi_n)$, we can simply do:

$$\begin{aligned} & J(\pi_{n+1}) - J(\pi_n) = J(\pi_{n+1}) - J(\eta_n) + J(\eta_n) - J(\pi_n) \\ & \leq \frac{\beta\varepsilon}{(1-\gamma)^2} - \frac{|\mathbb{A}_n(\pi_{n+1})|}{1-\gamma} - \Delta(\alpha). \end{aligned} \tag{39}$$

A.4. Proof of Theorem 3.3

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

$$\mathbb{A}_{\pi_n}(\pi_{n+1}) = \mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{a\sim\pi_{n+1}(\cdot|s)}[A^{\eta_n}(s,a)]] \leq \mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{s\sim\pi'_{n+1}}[A^{\eta_n}(s,a)]], \tag{40}$$

due to the fact that π_{n+1} is the optimal solution of the optimization problem shown in Eq. 4 subject to the trust region constraint. Hence if $\mathbb{A}_\pi(\pi_{n+1}) \geq -\xi$, we must have $\mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{s\sim\pi'_{n+1}}[A^{\eta_n}(s,a)]] \geq -\xi$, which means that:

$$\begin{aligned} & \mathbb{E}_{s\sim d_{\pi_n}}\left[(1-\beta)\mathbb{E}_{s\sim d_{\pi_n}} A^{\eta_n}(s,a) + \beta\mathbb{E}_{s\sim d_{\pi_n^*}} A^{\eta_n}(s,a)\right] \\ & = (1-\beta)(1-\gamma)(J(\pi_n) - J(\eta_n)) + \beta\mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{a\sim\pi_n^*} A^{\eta_n}(s,a)] \geq -\xi, \\ & \Rightarrow \mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{a\sim\pi_n^*} A^{\eta_n}(s,a)] \geq -\frac{\xi}{\beta} - \frac{1-\beta}{\beta}(1-\gamma)\Delta(\alpha) \geq -\frac{\xi}{\beta} - \frac{1-\gamma}{\beta}\Delta(\alpha). \end{aligned} \tag{41}$$

Recall the realizable assumption: $\mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{a\sim\pi_n^*} A^{\eta_n}(s,a)] = \mathbb{E}_{s\sim d_{\pi_n}}[\min_a A^{\eta_n}(s,a)]$, we have:

$$\begin{aligned} & -\frac{\xi}{\beta} - \frac{1-\gamma}{\beta}\Delta(\alpha) \leq \sum_s d_{\pi_n}(s) \min_a A^{\eta_n}(s,a) = \sum_s \frac{d_{\pi_n}(s)}{d_{\pi^*}(s)} d_{\pi^*}(s) \min_a A^{\eta_n}(s,a) \\ & \leq \min_s \left(\frac{d_{\pi_n}(s)}{d_{\pi^*}(s)}\right) \sum_s d_{\pi^*}(s) \min_a A^{\eta_n}(s,a) \\ & \leq \min_s \left(\frac{d_{\pi_n}(s)}{d_{\pi^*}(s)}\right) \sum_s d_{\pi^*}(s) \sum_a \pi^*(a|s) A^{\eta_n}(s,a) \\ & = \min_s \left(\frac{d_{\pi_n}(s)}{d_{\pi^*}(s)}\right) (1-\gamma)(J(\pi^*) - J(\eta_n)). \end{aligned} \tag{42}$$

Rearrange, we get:

$$\begin{aligned} J(\eta_n) - J(\pi^*) & \leq \left(\max_s \left(\frac{d_{\pi^*}(s)}{d_{\pi_n}(s)}\right)\right) \left(\frac{\xi}{\beta(1-\gamma)} + \frac{\Delta(\alpha)}{\beta}\right) \\ & \leq \left(\max_s \left(\frac{d_{\pi^*}(s)}{\rho(s)}\right)\right) \left(\frac{\xi}{\beta(1-\gamma)^2} + \frac{\xi}{\beta(1-\gamma)}\right) \end{aligned} \tag{43}$$

A.5. Analysis on Using DAgger for Updating π_n

To analyze the update of π using DAgger, we consider deterministic policy here: we assume π_n and η are both deterministic and the action space $\mathcal{A}$ is discrete. We consider the following update procedure for π:

$$\begin{aligned} & \min_{\pi\in\Pi} \mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi(\cdot|s)} \mathbb{1}(a \neq \arg\min_a A^{\eta_n}(s,a))\right], \\ & \quad s.t., \mathbb{E}_{s\sim d_{\pi_n}}[\|\pi(\cdot|s) - \pi_n(\cdot|s)\|_1] \leq \beta. \end{aligned} \tag{44}$$

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" η_n, running the above update step with respect to π can be regarded as running online gradient descent with a local metric defined by the trust-region constraint. Recall that η_n may from a different policy class than Π.

Assume that we learn a policy π_{n+1} that achieves ϵ_n prediction error:

$$\mathbb{E}_{s\sim d_{\pi_n}}\left[\mathbb{E}_{a\sim\pi_{n+1}(\cdot|s)}\left[\mathbb{1}(a \neq \arg\min_a A^{\eta_n}(s,a))\right]\right] \leq \epsilon_n. \tag{45}$$

Namely we assume that we learn a policy π_{n+1} such that the average probability of mismatch to η_n is at most ϵ_n.

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

$$\|d_{\pi_{n+1}} - d_{\pi_n}\|_1 \leq \frac{2\beta}{1-\gamma}. \tag{46}$$

Applying PDL, we have:

$$\begin{aligned}
&(1-\gamma)(J(\pi_{n+1}) - J(\eta_n)) = \mathbb{E}_{s\sim d_{\pi_{n+1}}}[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]] \\
&\leq \mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{a\sim\pi_{n+1}}[A^{\eta_n}(s,a)]] + \frac{2\beta\varepsilon}{1-\gamma} \\
&= \mathbb{E}_{s\sim d_{\pi_n}}[\sum_{a\neq\arg\min_a A^{\eta_n}(s,a)} \pi(a|s)A^{\eta_n}(s,a)] + \frac{2\beta\varepsilon}{1-\gamma} \\
&\leq (\max_{s,a} A^{\eta_n}(s,a))\mathbb{E}_{s\sim d_{\pi_n}}[\mathbb{E}_{a\sim\pi_{n+1}}\mathbb{1}(a \neq \arg\min_a A^{\eta_n}(s,a))] + \frac{2\beta\varepsilon}{1-\gamma} \\
&\leq \varepsilon'\epsilon_n + \frac{2\beta\varepsilon}{1-\gamma},
\end{aligned} \tag{47}$$

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

$$J(\pi_{n+1}) - J(\pi_n) \leq \frac{2\beta\epsilon}{(1-\gamma)^2} + \frac{\varepsilon'\epsilon_n}{1-\gamma} - \Delta(\alpha). \tag{48}$$

Now we can compare the above upper bound to the upper bound shown in Theorem 3.2. 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 π_{n+1} over π_n only consists of the improvement from the local optimal control $\Delta(\alpha)$. While in theorem 3.2, under the same assumption, except for $\Delta(\alpha)$, the improvement of π_{n+1} over π_n has an extra term $\frac{|\mathbb{A}_n(\pi_{n+1})|}{1-\gamma}$, which basically indicates that we learn a policy π_{n+1} that is one-step deviation improved over η_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" η_n (i.e., $\epsilon_n = 0$).

B. Missing Experiment Details

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

We follow (Scherrer, 2014) 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 maintain a tabular representation $\hat{P} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|\times|\mathcal{S}|}$, where each entry $P_{i,j,k}$ records the number of visits of the state-action-next state triple. We represent η as a 2d matrix $\eta \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$, 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 $\phi(s) \in \mathbb{R}^{d_s}$ (d_s is the dimension of feature space, which is $\log_2(|\mathcal{S}|)$ in our setup). Hence the reactive policy π_n sits in low dimension 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. 4) by Cost-Sensitive One Against All (CSOAA) classification technique. Specifically, given a set of states $\{s_i\}_i$, and a cost vector $\{A^{\eta_n}(s_i,\cdot) \in \mathbb{R}^{|\mathcal{A}|}\}$, we train a linear regressor $\hat{W} \in \mathbb{R}^{|\mathcal{A}|\times d_s}$ to predict the cost vector: $\hat{W}\phi(s) \approx \mathcal{A}^{\eta_n}(s,\cdot)$. Then π_n^* in Eq. 6 is just a classifier that predicts action $\arg\min_i(\hat{W}s)[i]$ corresponding to the smallest predicted cost. We then combine π_n^* with the previous policies as shown in Eq. 6 to make sure π_{n+1} satisfies the trust region constraint in Eq. 5.

For CPI, we estimate $A^{\pi_n}(s,a)$ 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. For our method, we alternately compute η_n using VI with the new cost shown in Eq. 20 and $\hat{P}$, and update the Lagrange multiplier μ, under convergence. Hence the only difference between our approach and CPI here is simply that we use A^{η_n} while CPI uses A^{π_n}.

Our results indicates that using A^{η_n} converges much faster than using A^{π_n}, though computing η_n is much more time consuming than computing A^{π_n}. But again we emphasize that computing η_n doesn't require extra samples. For real large discrete MDPs, we can easily plug in approximate VI techniques such as (Gorodetsky et al., 2015) to significantly speed up computing η_n.

B.2. Details for Updating Lagrange Multiplier μ

Though running gradient ascent on μ 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 (Finn et al., 2016). We set up $\mu_{\min}$ and $\mu_{\max}$. Starting from $\mu = \mu_{\min}$, we fix μ and compute η using the new cost c' as shown in Eq. 20 under the local dynamics $\hat{P}$ using LQR. We then compare $\mathbb{E}_{s\sim\mu_n} D_{KL}(\eta(\cdot|s), \pi_n(\cdot|s))$ to α. If η violates the constraint, i.e., $\mathbb{E}_{s\sim\mu_n} D_{KL}(\eta(\cdot|s), \pi_n(\cdot|s)) > \alpha$, then it means that μ is too small. In this case, we set $\mu_{\min} = \mu$, and compute new μ as $\mu = \min(\sqrt{\mu_{\min}\mu_{\max}}, 10\mu_{\min})$; On the other hand, if η satisfies the KL constraint, i.e, μ is too big, we set $\mu_{\max} = \mu$, and compute new μ as $\mu = \max(\sqrt{\mu_{\min}\mu_{\max}}, 0.1\mu_{\max})$. We early terminate the process once we find η such that $0.9\alpha \leq \mathbb{E}_{s\sim\mu_n} D_{KL}(\eta(\cdot|s), \pi_n(\cdot|s)) \leq 1.1\alpha$. We then store the most recent Lagrange multiplier μ which will be used as warm start of μ for the next iteration.

B.3. Details on 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. We will provide link to our implementation here for the final version.