# Inverse Reinforcement Learning without Reinforcement Learning

**Gokul Swamy** [1] **David Wu** [2] **Sanjiban Choudhury** [2 3] **J. Andrew Bagnell** [3 1] **Zhiwei Steven Wu** [1]

## Abstract

Inverse Reinforcement Learning (IRL) is a powerful set of techniques for imitation learning that aims to learn a reward function that rationalizes expert demonstrations. Unfortunately, traditional IRL methods suffer from a computational weakness: they require repeatedly solving a hard reinforcement learning (RL) problem as a subroutine. This is counter-intuitive from the viewpoint of reductions: we have reduced the *easier* problem of imitation learning to repeatedly solving the *harder* problem of RL. Another thread of work has proved that access to the side-information of the distribution of states where a strong policy spends time can dramatically reduce the sample and computational complexities of solving an RL problem. In this work, we demonstrate for the first time a more informed imitation learning reduction where we utilize the state distribution of the expert to alleviate the global exploration component of the RL subroutine, providing an *exponential* speedup in theory. In practice, we find that we are able to significantly speed up the prior art on continuous control tasks.

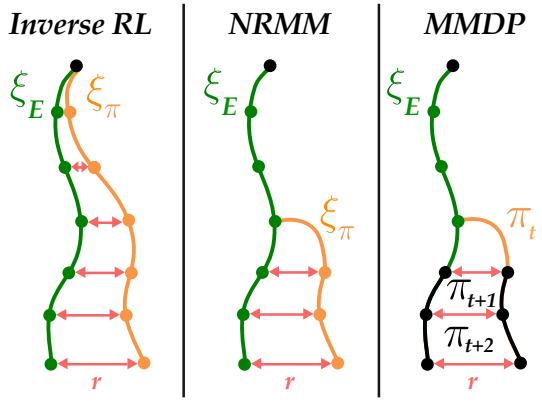



Figure 1: Traditional Inverse RL methods (left) repeatedly solve RL problems with adversarially chosen rewards in their inner loop which can be rather computationally expensive. We introduce two *exponentially* faster methods for IRL. `NRMM` (No-Regret Moment Matching, center) resets the learner to states from the expert demonstrations before comparing trajectory suffixes. `MMDP` (Moment Matching by Dynamic Programming, right) optimizes a sequence of policies backwards in time. Both methods avoid solving the global exploration problem inherent in RL.

## 1. Introduction

Inverse Reinforcement Learning (IRL), also known as Inverse Optimal Control (Kalman, 1964; Bagnell, 2015) or Structural Estimation (Rust, 1994), is the problem of finding a reward function that *rationalizes* (i.e. makes optimal) demonstrated behavior. Such approaches build on the lengthy history of trying to understand intelligent behavior (Muybridge, 1887) as approximate optimization of some cost function (Wolpert et al., 1995). While economists (Rust, 1994) and cognitive scientists (Baker et al., 2009) are often interested in analyzing the recovered reward function, it is more common in machine learning to view IRL algorithms as methods to *imitate* (Ziebart et al., 2008a) or *forecast* (Kitani et al., 2012) expert behavior.

There are three key benefits to the IRL approach to imitation. The first is *policy space structuring*: effectively, IRL reduces our (often large) policy class to just those policies that are (approximately) optimal under some member of our (relatively small) reward function class. The second is *transfer across problems*: for many practical applications (e.g. robotics (Silver et al., 2010; Ratliff et al., 2009; Kolter et al., 2008; Ng et al., 2006; Zucker et al., 2011), computer vision (Kitani et al., 2012), and human-computer interaction (Ziebart et al., 2008b; 2012)), one is able to learn a *single* reward function across multiple instances and then use it to forecast or imitate expert behavior in new problems that arise at test time. As Ng et al. (2000) put it, "the entire field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy is the most succinct, robust, and *transferable* definition of the task" (italics ours). The third is *robustness to compounding errors*:

[1]Carnegie Mellon University [2]Cornell University [3]Aurora Innovation. Correspondence to: Gokul Swamy <gswamy@cmu.edu>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

as IRL methods involve the learner performing rollouts in the environment, they cannot end up in states they didn't expect to at test time and therefore will not suffer from compounding errors (Swamy et al., 2021). Taken together, these three reasons help explain why IRL methods continue to provide state-of-the-art results on challenging imitation learning problems (e.g. in autonomous driving (Bronstein et al., 2022; Igl et al., 2022; Vinitsky et al., 2022)).

The most widely used approaches to IRL (Ziebart et al., 2008a; Ho and Ermon, 2016) are fundamentally game-theoretic. An RL algorithm *generates* trajectories by optimizing (i.e. *decoding*) the current reward function. In response, a reward function selector picks a new reward function that *discriminates* between learner and expert trajectories. As pointed out by Finn et al. (2016), the IRL setup generalizes a GAN (Goodfellow et al., 2020) with a dynamics model in the generation stem. More specifically, if one looks at the typical structure of an IRL algorithm, one performs the decoding-via-RL operation repeatedly in an *inner loop*, tweaking the current estimate of the reward function in the *outer loop* to produce behavior that more closely resembles that of the expert.

For some problems, highly optimized planners (Ratliff et al., 2009) or optimal controller synthesis procedures (Levine and Koltun, 2013) allow an efficient implementation of this inner loop. More generally however, one might want to tackle problems that don't have efficient algorithms to decode behavior and therefore be forced to rely on sample-based RL algorithms. Unfortunately, this can make each inner loop iteration quite inefficient (both in terms of computational and sample efficiency) as it requires solving the *global* exploration problem inherent in RL. From the lens of reductions (Beygelzimer et al., 2009), such an approach is counter-intuitive as we've turned the relatively easy problem of imitating an expert into the repeated solving of the hard problem of RL.

Prior work (Kakade and Langford, 2002; Bagnell et al., 2003; Ross et al., 2010) has shown that access to a good *exploration distribution* (i.e. the states where a strong policy spends much of its time) can dramatically reduce the complexity of RL as the learner doesn't have to explore for as long: knowing a set of waypoints along the shortest path through a maze should speed up your attempt to solve it. In the imitation learning setup, we have access to just such a distribution: *the expert's visitation distribution*. We propose to use this knowledge to speed up the RL subroutine of IRL.

Our key insight is that ***expert demonstrations can dramatically improve the efficiency of the policy optimization subroutine of IRL***. Critically, directly applying this insight to prior IRL algorithms with an RL inner loop provably fails to learn good behavior, as we discuss further below. Instead, ***we derive a new flavor of IRL algorithms that perform policy synthesis in the outer loop, eliding this concern***.

More explicitly, our contributions are as follows:

**1. We derive two algorithms that reset the learner to states from the expert visitation distribution for more efficient IRL.** `MMDP` (Moment Matching by Dynamic Programming) produces a sequence of policies. `NRMM` (No-Regret Moment Matching) produces a single, stationary policy and has best-response and no-regret variants. Further, we prove that the natural idea to use RL algorithms that leverage expert resets in the *inner* loop of game-theoretic IRL fails to recover a policy competitive with the expert's.

**2. We discuss the statistical complexity of expert resets.** We prove that in the worst case, traditional IRL algorithms take an *exponential* number of interactions (in the horizon of the problem) to learn a policy competitive with the expert. In contrast, we prove that our algorithms require only *polynomial* interactions per iteration to learn policies competitive with the expert.

**3. We discuss the performance implications of expert resets.** We show that in the worst case, neither `MMDP` nor `NRMM` can avoid a quadratic compounding of errors with respect to the horizon.

**4. We derive a practical meta-algorithm that achieves the best of both.** We propose `FILTER` (Fast Inverted Loop Training via Expert Resets) which interpolates between traditional IRL and our own approaches via mixing expert resets with standard resets. This allows use to ease the exploration burden on the learner while mitigating compounding errors. We implement two variants of `FILTER` on continuous control tasks and find it is more efficient than standard IRL.

We begin with a discussion of related work.

## 2. Related Work

Both of our algorithms build upon prior work in utilizing strong exploration distributions in the reinforcement learning context (Kakade and Langford, 2002). In a sense, we lift these insights to the imitation learning context. `MMDP` can be seen as the moment-matching version of the PSDP (Policy Search by Dynamic Programming) algorithm of Bagnell et al. (2003). `NRMM` calls the NRPI (No-Regret Policy Iteration) algorithm of Ross et al. (2010) in each iteration. Recent work by Uchendu et al. (2022) has confirmed that PSDP and NRPI continue to provide strong computational benefits with modern training algorithms and architectures, boding well for their application to IRL.

Our work is also related to recent developments in the theory of policy gradient algorithms by Agarwal et al. (2021a), in that we also assume access to a reset distribution that covers

**Algorithm 1** `IRL` (Dual Version, Ziebart et al. (2008a))

**Input:** Demos. $\mathcal{D}_E$, Policy class $\Pi$, Reward class $\mathcal{F}_r$
**Output:** Trained policy $\pi$
Initialize $f_1 \in \mathcal{F}_r$
**for** $i$ in $1 \dots N$ **do**
  $\pi_i \leftarrow \texttt{MaxEntRL}(r = -f_i)$
  `# use any no-regret algo. to pick f`
  $f_{i+1} \leftarrow \underset{f \in \mathcal{F}_r}{\text{argmax}} J(\pi_E, f) - J(\text{Unif}(\pi_{1:i}), f) + R(f)$
**end for**
**Return** $\pi_i$ with the lowest validation error.

**Algorithm 2** `IRL` (Primal Version, Ho and Ermon (2016))

**Input:** Demos. $\mathcal{D}_E$, Policy class $\Pi$, Reward class $\mathcal{F}_r$
**Output:** Trained policy $\pi$
Initialize $f_1 \in \mathcal{F}_r$
**for** $i$ in $1 \dots N$ **do**
  `# use any no-regret algo. to pick pi`
  $\pi_i \leftarrow \texttt{MaxEntRL}(r = -\frac{1}{i}\sum_{j=1}^{i} f_j)$
  $f_{i+1} \leftarrow \underset{f \in \mathcal{F}_r}{\text{argmax}} J(\pi_E, f) - J(\pi_i, f)$
**end for**
**Return** $\pi_i$ with the lowest validation error.

the visitation distribution of the policy we compare the learner's to. While they compare to the optimal policy, we compare to the expert, as we are focused IRL.

Swamy et al. (2022) also consider sample efficiency in IRL, but focus on making the most out of a finite set of expert demonstrations, rather than solving the moment-matching problem with as few learner-environment interactions as possible. Our work is therefore complementary to theirs.

Perhaps the most similar algorithm to `MMDP` is the FAIL algorithm of Sun et al. (2019). While both algorithms solve a sequence of moment-matching games, they differ in several key ways. Perhaps most obviously, FAIL is solving the sequence of games forward in time while `MMDP` is solving them backwards in time. This makes it straightforward to mix `MMDP` with value-based reinforcement learning (which also uses backwards-in-time dynamic programming), while it is not apparent how to do so for FAIL.

An alternative way to use expert demonstrations in policy optimization is via regularization towards a trained behavioral cloning policy (Jacob et al., 2022). The benefits of such a technique are quite problem-specific (e.g. such regularization could introduce compounding errors where none would exist otherwise). However, on problems where such regularization is helpful, it can easily be combined with the improved efficiency our techniques provide.

Another line of work attempts to improve the efficiency of IRL algorithms by learning $Q$ functions and then differencing them across sequential states to extract a reward function, eliding the need for an inner loop (Dvijotham and Todorov, 2010; Garg et al., 2021). While reward functions don't include information about the dynamics of the environment, $Q$ functions do. This means that $Q$ function-based approaches to IRL need to spend computation and samples to learn the environment's dynamics, only to immediately filter them out, which seems rather inefficient and can introduce other errors from the secondary regression. Furthermore, while one might be able to apply traditional IRL methods on datasets collected from diverse agents solving tasks with similar goals (Silver et al., 2010; Ratliff et al., 2009; Kolter et al., 2008; Ng et al., 2006; Zucker et al., 2011; Ziebart et al., 2008b; 2012), it isn't clear that $Q$ function-based approaches would output consistent estimates of the expert's reward function if dynamics differ across environments.

## 3. Expert Resets in Inverse RL

We utilize the moment-matching framework of Swamy et al. (2021) to prove performance bounds for our algorithms. Our results allow one to speed up *any* member of the broad reward-moment-matching phylum of their taxonomy that uses RL (e.g. MaxEnt IRL (Ziebart et al., 2008a), GAIL (Ho and Ermon, 2016), SQIL (Reddy et al., 2019)).

### 3.1. Inverse RL as (Inefficient) Game Solving

Consider a finite-horizon Markov Decision Process (MDP) (Puterman, 2014) parameterized by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, T \rangle$ where $\mathcal{S}, \mathcal{A}$ are the state and action spaces, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition operator, $r : \mathcal{S} \times \mathcal{A} \to [-1, 1]$ is the reward function, and $T$ is the horizon. In the inverse RL setup, we see trajectories generated by an expert policy $\pi^E : \mathcal{S} \to \Delta(\mathcal{A})$, but do not know the reward function. Our goal is to nevertheless learn a policy that performs as well as the expert's, no matter the true reward function.

We solve the IRL problem via equilibrium computation between a policy player and an adversary that tries to pick out differences between expert and learner policies along certain moments (i.e. potential components of the reward function) (Swamy et al., 2021). More formally, we optimize over (time-varying) policies $\pi = \{\pi_1, \dots, \pi_T\}$, with $\pi_t : \mathcal{S} \to \Delta(\mathcal{A}) \in \Pi$ and reward functions $f : \mathcal{S} \times \mathcal{A} \to [-1, 1] \in \mathcal{F}_r$. For simplicity, we assume that our strategy spaces ($\Pi$ and $\mathcal{F}_r$) are convex and compact, that $\mathcal{F}_r$ is closed under negation, and that $r \in \mathcal{F}_r, \pi_E \in \Pi$. [1] We solve (i.e. compute an approximate Nash equilibrium) of the two-

[1] If we do not assume realizability, we would get the analogous agnostic bounds throughout the following sections.

**Algorithm 3** MMDP (Moment Matching by Dynamic Programming)

**Input:** Sequence of expert visitation distributions $\rho_E^1 \dots \rho_E^T$, Policy class $\Pi$, Reward class $\mathcal{F}_r$
**Output:** Sequence of trained policies $\pi = \pi^{1:T}$
**for** $t$ in $T \dots 1$ **do**
  Set $\mathcal{D}_t = \{\}$
  **for** $j = 1$ to $M$ **do**
    Sample a random start state $s_t \sim \rho_E^t$.
    Execute a random action $a_t \sim \text{Unif}(\mathcal{A})$ in $s_t$.
    Follow $\pi^{t+1:T}$ until the end of the horizon.
    $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \{(s_t, a_t, s_{t+1:T}, a_{t+1:T})\}$
  **end for**
  Approximately solve moment-matching game:

$$\pi^t \leftarrow \arg\min_{\pi \in \Pi} \max_{f \in \mathcal{F}_r} \frac{1}{T} \left( \mathbb{E}_{\substack{s_t \sim \rho_E^t, \\ a_t \sim \pi(s_t)}} \left[ \mathbb{E}_{\mathcal{D}_t | s_t, a_t} \left[ \sum_{\tau=t}^{T} f(s_\tau, a_\tau) \right] \right] - \mathbb{E}_{\substack{s_t \sim \rho_E^t, \\ a_t \sim \rho_E^t(s_t)}} \left[ \mathbb{E}_{\mathcal{D}_t | s_t, a_t} \left[ \sum_{\tau=t}^{T} f(s_\tau, a_\tau) \right] \right] \right) \quad (1)$$

**end for**
**Return** $\pi^{1:T}$.

player zero sum game

$$\min_{\pi \in \Pi} \max_{f \in \mathcal{F}_r} J(\pi_E, f) - J(\pi, f), \quad (2)$$

where $J(\pi, f) = \mathbb{E}_{\xi \sim \pi}[\sum_{t=0}^{T} f(s_t, a_t)]$ denotes the value of policy $\pi$ under reward function $f$.

Swamy et al. (2021) describe two different classes of strategies for equilibrium computation: *primal*, where the policy player follows a no-regret strategy against a best-response discriminative player and *dual*, where the discriminative player follows a no-regret strategy against a best-response policy player. Most IRL algorithms are dual (e.g. MaxEnt IRL (Ziebart et al., 2008a) or LEARCH (Ratliff et al., 2009)) but there do exist primal approaches (e.g. GAIL (Ho and Ermon, 2016)). For both classes of strategies, a best-response corresponds to an inner loop iteration, while a no-regret step corresponds to an outer loop iteration.

For the policy player, a best-response consists of solving the RL problem under the current adversarially chosen reward function, i.e.

$$\pi_{i+1} = \operatorname*{argmax}_{\pi \in \Pi} J(\pi, f_i) + H(\pi), \quad (3)$$

while a no-regret step consists of running *any* no-regret online learning algorithm over the history of rewards, [2] e.g.

$$\pi_{i+1} = \operatorname*{argmax}_{\pi \in \Pi} J(\pi, \frac{1}{i} \sum_{j=0}^{i} f_j) + H(\pi), \quad (4)$$

[2]We write down a specific no-regret algorithm here (Follow the Regularized Leader) but one could use any other (e.g. Multiplicative Weights (Arora et al., 2012) or Online Gradient Descent (Zinkevich, 2003)) and have similar guarantees.

where $H(\pi)$ denotes the entropy of the policy. See Algorithms 1 and 2 for psuedocode, with $R(f)$ being a strongly convex regularizer.

In both cases, one is solving a full RL problem at each iteration. This means that in the worst case, one pays exponentially in the horizon at each iteration (Kakade, 2003):

**Theorem 3.1.** ***Inverse RL Sample Complexity:*** *For Algorithms 1 and 2, there exists an MDP, $\pi_E$, $\Pi$, and $\mathcal{F}_r$ such that returning a policy $\pi$ which satisfies $J(\pi_E, r) - J(\pi, r) \leq 0.5 V_{max}$ requires $\Omega(|\mathcal{A}|^T)$ interactions with the environment, where $V_{max}$ is the value of the optimal policy.*

We now discuss how we can utilize the (already known from the demonstrations) expert's visitation distribution to solve RL problems more efficiently.

### 3.2. Method 1: Dynamic Programming

Dynamic programming in the form of the Bellman Equation forms the basis of $Q$-learning based approaches to RL: one "backs-up" $Q$ values backwards-in-time, selecting actions based on the sum of the reward at the current timestep and the already computed value of the next state. More generally however, one can back-up *policies* rather than just $Q$-values, as in the Policy Search by Dynamic Programming (PSDP) algorithm of Bagnell et al. (2003). Given some roll-in distribution $\nu$, the algorithm draws states from timestep $T$ and selects a policy

$$\pi_T = \operatorname*{argmax}_{\pi \in \Pi} \mathbb{E}_{s \sim \nu^T}[r(s, \pi(s))]. \quad (5)$$

Then, holding this policy fixed, the algorithm draws states from the roll-in distribution at timestep $T-1$ and selects a

policy for timestep $T-1$ that maximizes reward over the horizon,

$$\pi_{T-1} = \operatorname*{argmax}_{\pi \in \Pi} \mathbb{E}_{s \sim \nu^{T-1}}[r(s, \pi(s)) + r(s', \pi_T(s'))], \quad (6)$$

where $s'$ denotes a successor state. This induction proceeds backwards in time until one reaches the first timestep, at which point a sequence of policies $\pi_{1:T}$ is output. Notice that at each step of this algorithm, we are solving a single-step classification problem. So, instead of the exponential-in-the-horizon complexity one must pay (in hard instances) for RL, one pays only *quadratically* in the horizon.

The careful reader will notice that PSDP requires a reward function. Two strategies come to mind for adversarially picking one for IRL. The first is to choose a reward for each *timestep* (i.e. each $t \in [T]$) of PSDP. The second is to run PSDP to completion (i.e. solve for all $T$ policies) and then pick a new reward in an outer loop. While the latter strategy is more similar to standard IRL methods, we prove in a later section that such approaches can provably fail to match expert behavior. We therefore propose using the former strategy, for which we can provide strong guarantees. We call the resulting algorithm MMDP: *Moment Matching by Dynamic Programming* and outline the procedure in Algorithm 3. Throughout our analysis, we define optimization error $\epsilon_t$ as the value when $\pi_t$ is plugged into Eq. (1). Like PSDP, MMDP avoids the exponential sample complexity of RL.

**Lemma 3.2.** ***MMDP Sample Complexity:*** *Let $\epsilon > 0$. At iteration $t$, MMDP requires at most*

$$O\left(\log\left(\frac{|\Pi||\mathcal{F}_r|}{\delta}\right)\frac{T^3|\mathcal{A}|^2}{\epsilon^2}\right)$$

*interactions with the environment to, w.p. $\geq 1-\delta$, produce a policy $\pi_t$ with optimization error $\epsilon_t \leq \epsilon$ (Eq. 1).*

MMDP performs $T$ iterations, giving us an overall complexity that is still polynomial in the relevant quantities. [3] We prove the following performance bound on the policies produced by MMDP in Appendix A:

**Theorem 3.3.** ***MMDP Upper Bound:*** *Let $\pi$ denote the sequence of policies returned by MMDP and $\bar{\epsilon} = \frac{1}{T}\sum_t^T \epsilon_t$, where $\epsilon_t$ denotes the optimization error of $\pi_t$ (Eq. 1). Then,*

$$J(\pi_E) - J(\pi) \leq \bar{\epsilon} T^2 \quad (7)$$

This bound tells us how training error $\bar{\epsilon}$ translates to our policy's test-time performance. The lower bound matches, making the above tight.

[3] For simplicity, we consider finite classes. One could instead use another complexity measure (e.g. Rademacher) that extends to classes with infinite elements (Sriperumbudur et al., 2009).

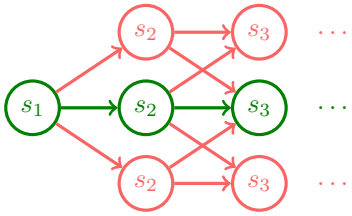



Figure 2: DANTE: A three-row MDP where at each timestep, the learner can move up, move down, or stay in the same row. The expert always stays in the center row. The goal is to stay in the top two rows.

**Theorem 3.4.** ***MMDP Lower Bound:*** *There exists an MDP, $\pi_E$ and sequence of policies $\pi$ with $\bar{\epsilon} = \frac{1}{T}\sum_t^T \epsilon_t$, where $\epsilon_t$ denotes the optimization error of $\pi_t$ (Eq. 1), such that*

$$J(\pi_E) - J(\pi) \geq \Omega(\bar{\epsilon} T^2) \quad (8)$$

Intuitively, a single mistake early on in an episode can put the learner in a different part of the state space than the expert, which can make the learned policy perform poorly. In short, MMDP is able to find a sequence of policies in polynomial time that perform at most $\bar{\epsilon}T^2$ worse than $\pi_E$.

**MMDP vs. Behavioral Cloning.** A natural question at this point might be: *what benefits does MMDP provide over a behavioral cloning baseline?* After all, behavioral cloning also produces policies that do no worse than $O(\epsilon T^2)$ compared to the expert and requires no environment interaction.

Consider a simplified variant of MMDP in which one doesn't perform rollouts and instead solves a game with a single-timestep payoff at each iteration. This entirely decouples the iterations as we no longer account for the actions of the future policies we have already computed. In effect, this is what purely offline behavioral cloning is doing.

The core issue with such an approach is that *it prevents the learner from distinguishing between mistakes that compound over the horizon and those that don't.* Consider, for example, the MDP depicted in Figure 2 where the goal is to stay in the top two rows. Assume policies $\pi_{3:T}$ go straight but $\pi_2$ goes down w.p. $\epsilon T$. Now, let's think about what would happen if we used BC or MMDP to pick $\pi_1$. Behavioral cloning would pick a policy that always goes straight, as doing so perfectly matches expert actions. This would lead to a performance gap of

$$J(\pi_E, r) - J(\{\pi_{BC}, \pi_{2:T}\}, r) = \epsilon T(T-1).$$

However, if we instead used MMDP to pick $\pi_1$, the rollouts with $\pi_{2:T}$ would reveal to the learner that it is better to go up on the first timestep so they still receive reward over the horizon, no matter what $\pi_2$ chooses. Thus, the learner

**Algorithm 4** `NRMM(BR)` (No-Regret Moment Matching: Best Response Variant)

**Input:** Sequence of expert visitation distributions $\rho_E^1 \dots \rho_E^T$, Policy class $\Pi$, Reward class $\mathcal{F}_r$
**Output:** Trained policy $\pi$
Set $\pi_0 \in \Pi$, $\mathcal{D} = \{\}$
**for** $i = 1$ to $N$ **do**
  Set $\mathcal{D}_{i-1} = \{\}$
  **for** $j = 1$ to $M$ **do**
    Sample random time $t \sim \text{Unif}([0, T])$ and start state $s_t \sim \rho_E^t$.
    Execute a random action $a_t \sim \text{Unif}(\mathcal{A})$ in $s_t$.
    Follow $\pi_{i-1}$ until the end of the horizon.
    $\mathcal{D}_{i-1} \leftarrow \mathcal{D}_{i-1} \cup \{(s_t, a_t, t, s_{t+1:T}, a_{t+1:T})\}$
  **end for**
  Let

$$L(\pi_{i-1}, f) = \mathbb{E}_{\xi \sim \pi_{i-1}} \left[ \sum_{t=0}^{T} f(s_t, a_t) \right] - \mathbb{E}_{\xi \sim \rho_E} \left[ \sum_{t=0}^{T} f(s_t, a_t) \right] \tag{9}$$

  Optimize $f_{i-1} \leftarrow \arg\max_{f \in \mathcal{F}_r} L(\pi_{i-1}, f)$. `# for NRMM(NR), optimize` $L(\text{Unif}(\pi_{1:i-1}), \cdot)$ `instead`
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, \hat{Q}_t = \sum_{\tau=t}^{T} f_{i-1}(s_\tau, a_\tau) | \text{tuple} \in \mathcal{D}_{i-1}\}$
  `# Run any no-regret algorithm on` $\mathcal{D}_{1:i-1}$ `to produce new` $\pi_i$`, e.g. FTRL:`
  Optimize

$$\pi_i \leftarrow \operatorname*{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi(s)}[\mathbb{E}[\hat{Q}_t | s_t = s, a_t = a]] - \alpha H(\pi). \tag{10}$$

**end for**
**Return** $\pi_i$ with lowest validation error.

would match expert performance, i.e.

$$J(\pi_E, r) - J(\{\pi_{\texttt{MMDP}}, \pi_{2:T}\}, r) = 0.$$

So, while in the worst case, BC and `MMDP` might both perform poorly (e.g. if the learner falls off a cliff and is stuck for the rest of the episode), we would expect that for a wide set of practical problems, knowledge of future choices would enable better performance over the horizon.

### 3.3. Method 2: No-Regret Moment Matching

For tasks with long horizons, learning a sequence of policies may be significantly more burdensome than learning just one. We now present an algorithm that outputs a single, stationary policy. Our approach is based on the No-Regret Policy Iteration (NRPI) algorithm of Ross et al. (2010). Instead of solving a sequence of optimization problems backwards in time like PSDP, NRPI picks a time to sample from the roll-in distribution uniformly at random, takes a random action, and then follows the previous policy $\pi_{i-1}$ for the rest of the episode. This gives it sample estimates of $Q^{\pi_{i-1}}$ on states from the roll-in distribution. To have a no-regret property, NRPI performs (regularized) greedy policy improvement using the *history* of such samples, i.e.

$$\pi_i = \operatorname*{argmin}_{\pi \in \Pi} \sum_{j=0}^{i-1} \mathbb{E}_{t \sim U[0,T], s \sim \eta^t}[Q^{\pi_j}(s, \pi(s))] - \alpha H(\pi).$$

Notice that rather than solving a global exploration problem, NRPI only focuses on picking the best action on states from the roll-in distribution, avoiding the exponential interaction complexity lower bound. NRPI can be seen as an analog of PSDP for stationary policies (Ross et al., 2010). As with PSDP, NRPI requires a reward function. We therefore choose one adversarially for IRL. We outline the full procedure in Algorithm 4. Intuitively, this algorithm is performing *primal* moment-matching with the learner's start state distribution being the expert's stationary distribution (i.e. Algorithm 2 or GAIL with expert resets).

Let $L(\pi, \mathcal{D}_i) = \mathbb{E}_{s \sim \rho_E, a \sim \pi(s)}[\mathbb{E}_{\mathcal{D}_i}[\hat{Q}_t | s_t = s, a_t = a]]$ denote the cost-sensitive classification loss of policy $\pi$ over dataset $\mathcal{D}_i$. We use the following regret measure in our analysis:

$$\epsilon_i = L(\pi_i, \mathcal{D}_i) - L(\pi^*, \mathcal{D}_i), \tag{11}$$

where $\pi^* = \text{argmin}_{\pi \in \Pi} \sum_i^N L(\pi, \mathcal{D}_i)$. Like `MMDP`, `NRMM` has polynomial time iterations.

**Lemma 3.5.** ***`NRMM` Sample Complexity:*** *Let $\epsilon > 0$. At iteration $i$, `NRMM` requires at most*

$$O\left( \log\left( \frac{|\Pi||\mathcal{F}_r|}{\delta} \right) \frac{T^3 |\mathcal{A}|^2}{\epsilon^2} \right)$$

*interactions with the environment to, w.p. $\geq 1 - \delta$, produce a policy $\pi_i$ with instantaneous regret $\epsilon_i \leq \epsilon$ (Eq. 11).*

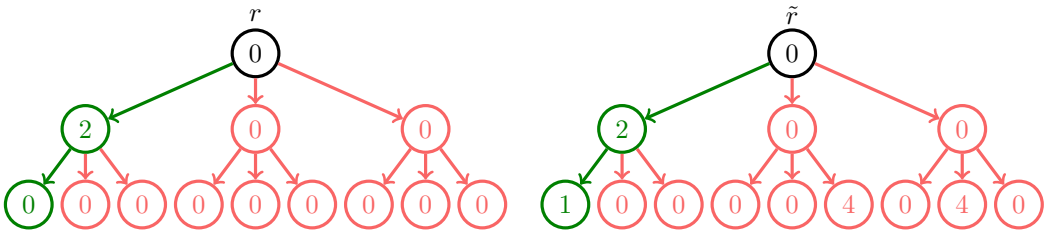



Figure 3: FORKED TREE: a tree-structured MDP with two rewards. The number on each node is the reward an agent gets for arriving at that state. Green nodes and arrows are the states / actions visited / taken by the expert, who always goes left.

However, unlike MMDP which always has $T$ outer-loop iterations, NRMM must be run until the average training error drops below some threshold on $\bar{\epsilon}$. While the particular number of iterations $N$ is a problem-specific quantity, the fact that the policy is selected by a no-regret algorithm tells us that, by definition,

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \epsilon_i = 0. \tag{12}$$

Thus, regardless of the desired $\bar{\epsilon}$, the outer loop will eventually terminate, with $\bar{\epsilon} \propto \frac{1}{\sqrt{N}}$ or $\bar{\epsilon} \propto \frac{\log(N)}{N}$ for a wide set of problems (Hazan, 2019), giving us poly-time bounds. There exist two variations of NRMM: one in which the adversary plays a best-response (i.e. differentiating between the current policy and expert demos – labeled as NRMM(BR)) and another in which the adversary follows a no-regret strategy (i.e. differentiating between replay buffer $\mathcal{D}$ and expert demos – labeled as NRMM(NR)). Both share similar policy performance guarantees (Appendix A).

**Theorem 3.6.** ***NRMM(BR) Upper Bound:*** *Let $\pi_1, \ldots, \pi_N$ denote the sequence policies computed by NRMM(BR) and $\bar{\epsilon} = \frac{1}{N}\sum_{i=1}^{N} \epsilon_i$ their average regret (Eq. 11). Then, $\exists \pi \in \{\pi_1, \ldots, \pi_N\}$ s.t.*

$$J(\pi_E) - J(\pi) \leq \bar{\epsilon} T^2 \tag{13}$$

When we use a no-regret algorithm to pick $f_i$ rather than a best response, we need to consider the instantaneous regrets of said algorithm. Let $f^* = \text{argmax}_{f\in\mathcal{F}_r} \sum_{i=1}^{N} L(\pi_i, f)$ and

$$\delta_i = L(\pi_i, f^*) - L(\pi_i, f_i), \tag{14}$$

where $L$ is as defined in Eq. (9). We can now give a performance guarantee as a function of $\epsilon_i$ and $\delta_i$.

**Theorem 3.7.** ***NRMM(NR) Upper Bound:*** *Let $\pi_1, \ldots, \pi_N$ and $f_1, \ldots, f_N$ denote the sequence policies and rewards computed by NRMM(NR) and $\bar{\epsilon} = \frac{1}{N}\sum_{i=1}^{N} \epsilon_i$, $\bar{\delta} = \frac{1}{N}\sum_{i=1}^{N} \delta_i$ their respective average regrets (Eqs. 11, 14). Then, the uniform mixture over policies $\bar{\pi}$ satisfies*

$$J(\pi_E) - J(\bar{\pi}) \leq (\bar{\epsilon} + \bar{\delta}) T^2 \tag{15}$$

These bounds are tight, via a similar construction to before.

**Theorem 3.8.** ***NRMM Lower Bound:*** *There exists an MDP, $\pi_E$ and $\pi$ with average training error $\bar{\epsilon}$ such that*

$$J(\pi_E) - J(\pi) \geq \Omega(\bar{\epsilon} T^2) \tag{16}$$

As NRMM also performs rollouts in the environment, our discussion on why MMDP is preferable to behavioral cloning also applies to NRMM.

We now highlight two nuances related to NRMM.

**Dual Algorithm.** As the policy is chosen via a no-regret algorithm, NRMM is a *primal* algorithm like Algorithm 2. A natural question at this point might be whether there is a *dual* algorithm of NRMM, where one uses NRPI in the inner loop to compute a best-response against an outer-loop no-regret adversary, akin to Algorithm 1.

Recall that NRPI only competes with policies that have similar visitation distributions to the expert (Ross et al., 2010). This is fine when it is selecting policies in the outer loop (as the expert policy is an equilibrium strategy), but not when it is the inner loop (as the expert policy might be quite far from the optimal policy for the adversarially chosen reward). We make this point with the example in Figure 3, which we analyze more formally in Appendix A.

**Theorem 3.9.** *There exists an MDP, $\pi_E$, $\Pi$, and $\mathcal{F}_r$ such that NRMM(BR), NRMM(NR), MMDP, and Algorithms 1 / 2 converge in a finite number of iterations to $\pi_E$, but the dual algorithm of NRMM never picks $\pi_E$ on any iteration.*

Notice that in Figure 3, the expert policy is not optimal for distractor reward $\tilde{r}$, even when starting from its own state distribution. Thus, if NRPI is passed $\tilde{r}$ in the inner loop of dual version of NRMM, it will perform an incorrect best response, preventing proper equilibrium computation. We also note that because the expert policy can be arbitrarily bad on a single distractor reward, game-solving techniques that require a uniform best-response approximation guarantee (Kakade et al., 2007) would have vacuous bounds.

**Discriminator Training.** We prove that the standard trajectory-level discriminator training usually performed in IRL (i.e. Eq. 9 in Algorithm 4) is lower variance than the suffix-level discriminator training one might think to perform based on the samples in replay buffer $\mathcal{D}$. We prove this point more formally in Appendix A.

## 4. Getting the Best of Both Worlds

In the preceding section, we derived two algorithms, MMDP and NRMM, which can compute policies that match expert behavior in polynomial time. However, in the worst case, both can produce policies that suffer from a quadratic compounding of errors with respect to the horizon. Traditional IRL approaches have complimentary strengths: they can suffer from exponential computation complexity but produce policies with a performance gap linear in the horizon. This begs the question: *can we get the best of both worlds?*

Consider a variation of NRMM where, with probability $\alpha$, we perform an expert reset, otherwise performing a standard rollout (i.e. $s_t \sim \rho^t_{\pi_{i-1}}$). By setting $\alpha = 1$, we unsurprisingly recover NRMM. However, if we set $\alpha = 0$, the per-round loss that is passed to the learner becomes

$$L(\pi, \mathcal{D}_i) = \mathbb{E}_{s\sim\rho_i, a\sim\pi(s)}[\mathbb{E}_{\mathcal{D}_i}[\hat{Q}_t | s_t = s, a_t = a]], \quad (17)$$

This is strikingly similar to the standard approximate policy improvement procedure (Sutton and Barto, 2018) with an adversarially chosen reward. Recall that in NRMM, we select our discriminator $f$ as in primal IRL (Algorithm 2). Put together, setting $\alpha = 0$ is effectively using an off-policy RL algorithm in the policy optimization component of Algorithm 2. One might therefore reasonably expect such an approach to inherit the exponential complexity and linear-in-the-horizon performance gap of standard IRL.

It is natural to consider annealing between these extremes by decaying $\alpha$ from 1 to 0 over outer-loop iterations. Intuitively, this allows the learner to quickly find a policy with quadratic errors before refining it to a policy with error linear in the horizon. Even more simply, one can interpolate with a fixed $\alpha = 0.5$ probability, reducing the exploration burden on the learner while mitigating compounding errors. We term such annealed / interpolated approaches FILTER: Fast Inverted Loop Training via Expert Resets. Defining $\bar{\epsilon}$ as in Eq. (11) and $\bar{\epsilon}_{RL}$ as

$$\bar{\epsilon}_{\text{RL}} = \frac{1}{NT}\sum_i^N (\max_{f_i \in \mathcal{F}_r} J(\pi_E, f_i) - J(\pi_i, f_i)), \quad (18)$$

(i.e. the errors on the expert and start state distributions) we can derive a performance bound for FILTER by taking the minimum over the NRMM and IRL bounds.

**Corollary 4.1.** ***FILTER Upper Bound:*** *Consider a set of policies $\{\pi_1, \ldots, \pi_N\}$ with errors $\bar{\epsilon}$ and $\bar{\epsilon}_{RL}$ (Eqs. (11), (18)). Then, we have that $\exists \pi \in \{\pi_1, \ldots, \pi_N\}$ s.t.*

$$J(\pi_E) - J(\pi) \leq \min\left(\bar{\epsilon}T^2, \bar{\epsilon}_{RL}T\right). \quad (19)$$

The expert reset probability $\alpha$ controls the trade-off or schedule of minimizing $\bar{\epsilon}$ ($\alpha \approx 1$) versus $\bar{\epsilon}_{\text{RL}}$ ($\alpha \approx 0$). Intuitively, FILTER inherits the transferability of the reward function across problems of IRL, has better robustness to inaccuracy in $\rho_E$ and compounding errors than NRMM, and is better able to handle recoverable situations than behavioral cloning.

Unfortunately, it is difficult to prove more about FILTER. This is because a learner's performance on a mixture of two distributions doesn't easily translate to a bound on their performance on either. Traditional approaches to deriving such a bound (e.g. as function of the $\mathcal{H}\Delta\mathcal{H}$ divergence (Ben-David et al., 2010)) produce vacuous bounds when applied to flexible hypothesis classes like neural networks. Similar difficulties have been encountered by others in the IRL community without resolution (Chang et al., 2015).

## 5. Experiments

We conduct experiments with the PyBullet Suite (Coumans and Bai, 2016). We train experts using RL and then present all learners with 25 expert demonstrations to remove small-data concerns. As a simple behavioral cloning baseline matches expert performance under these conditions (Swamy et al., 2021), we harden the problem by introducing randomization: with probability $p_{tremble}$, a random action gets executed in the environment rather than the one the policy chose. Our expert data is free from these corruptions. We also conduct experiments on the antmaze-large tasks from Fu et al. (2020), but with $p_{tremble} = 0$.

We compare 4 algorithms: FILTER(BR), FILTER(NR), MM (i.e. Algorithm 2, or, equivalently, FILTER(NR) with $\alpha = 0$), and behavioral cloning. See Appendix B for details. We do not implement MMDP as these tasks can all last for $T = 1000$ timesteps. We plot the performance of the policy as a function of the number of environment interactions used for policy optimization. [4] As recommended by Agarwal et al. (2021b), we plot a robust statistic (i.e. the interquartile mean). Standard errors are computed across 10 runs.

For our baseline moment-matching algorithm, we use a significantly improved version of GAIL (Ho and Ermon, 2016). Specifically, we switch from the Jensen-Shannon divergence to an integral probability metric (as recommended by Swamy et al. (2021)), use the more efficient Soft Actor

[4] In some implementations of algorithms like GAIL, trajectories from the policy's replay buffer are used for training the discriminator rather than trajectories sampled post-policy-update. For FILTER, as we may only observe suffixes when $\alpha > 0$, we need to separately sample whole trajectories post-policy-update. To make the comparison fair, we do this for MM as well.

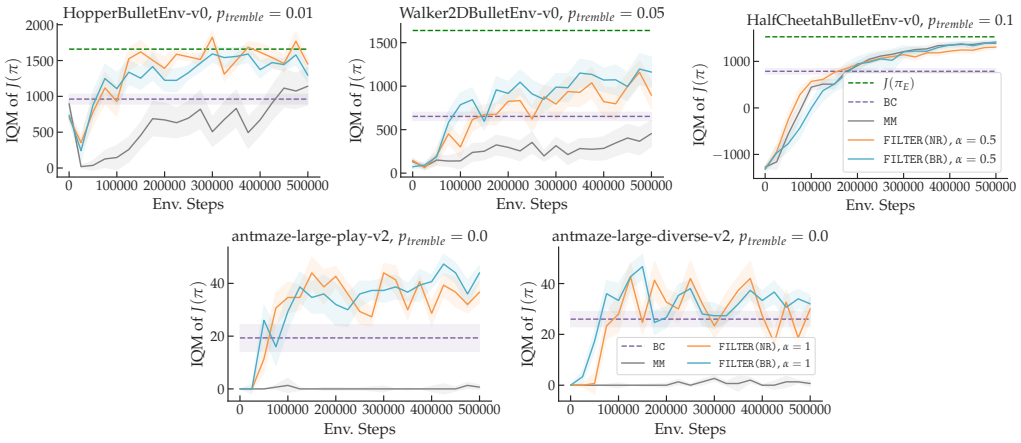



Figure 4: We see that both `FILTER(BR)` and `FILTER(NR)` out-performs standard `MM` and `BC` on 4 out of the 5 environments considered. Standard errors are computed across 10 seeds.

Critic (Haarnoja et al., 2018) or TD3+BC (Fujimoto and Gu, 2021) as our policy optimizers, add a gradient penalty to the discriminator (Gulrajani et al., 2017), and use Optimistic Mirror Descent (Daskalakis et al., 2017) to optimize both players for fast and last iterate convergence. See the appendix of Swamy et al. (2022) for an ablation of these changes. Taken together, these changes make our baseline a strong point of comparison, over which improvement is non-trivial.

In Figure 4, we see that `FILTER(BR)` and `FILTER(NR)` perform comparably and are significantly faster at finding strong policies than `MM` on 4/5 environments. We would recommend trying both variants when applying the algorithm in practice. To the best of our knowledge, the performance of `FILTER` on both variants of `antmaze` is the highest performance ever achieved by an algorithm that doesn't use any reward information. [5]

It is also interesting to consider the difference in results between the environments we consider. In the Bullet locomotion environments, we found that $\alpha = 0.5$ worked better than $\alpha = 1$. We hypothesize that this is because the learner is able to learn to connect their initial state to sampled expert states more easily. For locomotion tasks, this might correspond to learning to accelerate before matching the expert's gait. We tried a more complex annealing strategy but found that it did not outperform a fixed $\alpha = 0.5$. However, we believe that for other problems, the annealing strategy could perform better than a fixed $\alpha$.

For the AntMaze environments, we found that $\alpha = 1$ worked better than lower values. We hypothesize that this is because of the difficulty of exploration in a maze, for which expert resets can help a lot. In general, we would recommend that the harder exploration is in a problem, the higher $\alpha$ should be set.

We release the code we used for all of our experiments at **`https://github.com/gkswamy98/fast_irl`**. Of particular interest are the gym wrappers, which should be easily transferable to other implementations / IRL algorithms.

## 6. Discussion

In summary, we provide multiple algorithms for more sample efficient inverse reinforcement learning, both in theory and practice. Our key insight is speeding up policy optimization via resetting the learner to states from expert demonstrations. We emphasize that due to the reduction-based analysis we perform, one could apply this technique to an arbitrary primal inverse reinforcement learning algorithm and not just the GAIL-based approach we use in the experiments section. One interesting avenue for future work is developing an algorithm with stronger guarantees in the interpolated case – for example, one could imagine training two discriminators (one on trajectories from each start state distribution) and using the more pessimistic one as the reward function for the learner.

## 7. Acknowledgments

We thank Vasilis Syrgkanis for pointing out several typos in our proofs. ZSW is supported in part by the NSF FAI Award #1939606, a Google Faculty Research Award, a J.P. Morgan Faculty Award, a Facebook Research Award, an Okawa Foundation Research Grant, and a Mozilla Research Grant. GS is supported computationally by a GPU award from NVIDIA and emotionally by his family and friends.

[5] We note that the performance we report for behavioral cloning on these environments is significantly higher than what is usually reported in the literature – see Appendix B for details.

## References


Rudolf Emil Kalman. When is a linear control system optimal? 1964.

J Andrew Bagnell. An invitation to imitation. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Robotics Inst, 2015.

John Rust. Structural estimation of markov decision processes. *Handbook of econometrics*, 4:3081–3143, 1994.

Eadweard Muybridge. *Animal locomotion*, volume 534. Da Capo Press, 1887.

Daniel M Wolpert, Zoubin Ghahramani, and Michael I Jordan. An internal model for sensorimotor integration. *Science*, 269(5232):1880–1882, 1995.

Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113 (3):329–349, 2009.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008a.

Kris M Kitani, Brian D Ziebart, James Andrew Bagnell, and Martial Hebert. Activity forecasting. In *European conference on computer vision*, pages 201–214. Springer, 2012.

David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010.

Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. A control architecture for quadruped locomotion over rough terrain. In *2008 IEEE International Conference on Robotics and Automation*, pages 811–818. IEEE, 2008.

Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer, 2006.

Matt Zucker, Nathan Ratliff, Martin Stolle, Joel Chestnutt, J Andrew Bagnell, Christopher G Atkeson, and James Kuffner. Optimization and learning for rough terrain legged locomotion. *The International Journal of Robotics Research*, 30(2):175–191, 2011.

Brian D Ziebart, Andrew L Maas, Anind K Dey, and J Andrew Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 322–331, 2008b.

Brian Ziebart, Anind Dey, and J Andrew Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 1–10, 2012.

Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Gokul Swamy, Sanjiban Choudhury, J. Andrew Bagnell, and Zhiwei Steven Wu. Of moments and matching: A game-theoretic framework for closing the imitation gap, 2021. URL https://arxiv.org/abs/2103.03236.

Eli Bronstein, Mark Palatucci, Dominik Notz, Brandyn White, Alex Kuefler, Yiren Lu, Supratik Paul, Payam Nikdel, Paul Mougin, Hongge Chen, et al. Hierarchical model-based imitation learning for planning in autonomous driving. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8652–8659. IEEE, 2022.

Maximilian Igl, Daewoo Kim, Alex Kuefler, Paul Mougin, Punit Shah, Kyriacos Shiarlis, Dragomir Anguelov, Mark Palatucci, Brandyn White, and Shimon Whiteson. Symphony: Learning realistic and diverse agents for autonomous driving simulation. *arXiv preprint arXiv:2205.03195*, 2022.

Eugene Vinitsky, Nathan Lichtlé, Xiaomeng Yang, Brandon Amos, and Jakob Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. *arXiv preprint arXiv:2206.09889*, 2022.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016.

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

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

Alina Beygelzimer, John Langford, and Bianca Zadrozny. Tutorial summary: Reductions in machine learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1–1, 2009.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

James Bagnell, Sham M Kakade, Jeff Schneider, and Andrew Ng. Policy search by dynamic programming. *Advances in neural information processing systems*, 16, 2003.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2010. URL https://arxiv.org/abs/1011.0686.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. Jump-start reinforcement learning. *arXiv preprint arXiv:2204.02372*, 2022.

Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *J. Mach. Learn. Res.*, 22(98):1–76, 2021a.

Gokul Swamy, Nived Rajaraman, Matthew Peng, Sanjiban Choudhury, J Andrew Bagnell, Zhiwei Steven Wu, Jiantao Jiao, and Kannan Ramchandran. Minimax optimal online imitation learning via replay estimation. *arXiv preprint arXiv:2205.15397*, 2022.

Wen Sun, Anirudh Vemula, Byron Boots, and Drew Bagnell. Provably efficient imitation learning from observation alone. In *International conference on machine learning*, pages 6036–6045. PMLR, 2019.

Athul Paul Jacob, David J Wu, Gabriele Farina, Adam Lerer, Hengyuan Hu, Anton Bakhtin, Jacob Andreas, and Noam Brown. Modeling strong and human-like gameplay with kl-regularized search. In *International Conference on Machine Learning*, pages 9695–9728. PMLR, 2022.

Krishnamurthy Dvijotham and Emanuel Todorov. Inverse optimal control with linearly-solvable mdps. In *ICML*, 2010.

Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, 34:4028–4039, 2021.

Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

Bharath K Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, and Gert RG Lanckriet. On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*, 2009.

Elad Hazan. Introduction to online convex optimization, 2019. URL https://arxiv.org/abs/1909.05207.

Sham M Kakade, Adam Tauman Kalai, and Katrina Ligett. Playing games with approximation algorithms. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 546–555, 2007.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning*, pages 2058–2066. PMLR, 2015.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021b.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training gans with optimism. *arXiv preprint arXiv:1711.00141*, 2017.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3, 2019.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

# A. Proofs

## A.1. Sample Complexity Lemma

We follow standard techniques and include the proof here mostly for completeness.

**Lemma A.1.** *Consider $C$ deterministic functions of a random variable, each with range $R$. If we draw*

$$m \geq O\left(\log\left(\frac{C}{\delta}\right)\frac{R^2}{\epsilon^2}\right) \tag{20}$$

*samples, we have that with probability $\geq 1-\delta$, we will be able to estimate all $C$ population means within $\epsilon$ absolute error.*

*Proof.* Consider a bounded random variable $X$ with range $R$. A standard Hoeffding bound tells us that

$$P(|\frac{1}{m}\sum_{i=0}^{m} X_i - \mathbb{E}[X]| \geq \epsilon) \leq 2\exp(\frac{-2m\epsilon^2}{R^2}). \tag{21}$$

If we have $C$ such variables and want to be within $\epsilon$ of the population mean uniformly, a union bound tells us that we will do so with probability at least

$$1 - 2C\exp(\frac{-2m\epsilon^2}{R^2}). \tag{22}$$

If we want to satisfy this condition with probability at least $1-\delta$, simple algebra tells us that we must draw

$$m \geq O\left(\log\left(\frac{C}{\delta}\right)\frac{R^2}{\epsilon^2}\right) \tag{23}$$

samples. □

## A.2. Proof of Theorem 3.1

This construction is essentially a slight generalization of that of Kakade (2003) to the case with multiple reward functions.

*Proof.* Consider a tree-structured MDP with branching factor $|\mathcal{A}|$ and deterministic dynamics. The expert always takes the left-most action and therefore always ends up at the left-most node. Let $\mathcal{F}_r$ be the set of sparse reward functions that are 1 at a single leaf node and 0 everywhere else. Let $\Pi$ be the full set of deterministic policies (i.e. paths to a leaf node). Note that $|\mathcal{F}_r| = |\Pi| = |\mathcal{A}|^T$. Also note that $V_{max} = 1$ and that only one $\pi \in \Pi$ achieves nonzero reward under the true reward function, so one needs to find $\pi_E$ to satisfy the condition in the theorem statement.

Let us first analyze the dual version of IRL (Algorithm 1). At each iteration, the policy player solves a fresh RL problem with $r = -f \in \mathcal{F}_r$. As all $f \in \mathcal{F}_r$ are sparse, the learner needs to visit all nodes in the tree to find which one provides reward. As $|\mathcal{S}| \geq \Omega(|\mathcal{A}|^T)$, this must take at least $\Omega(|\mathcal{A}|^T)$ interactions with the environment.

We now analyze the primal version of IRL (Algorithm 2). While for $i > 1$ there could now exist multiple leaf nodes with reward under aggregate reward function $r = \frac{-1}{i}\sum_{j=1}^{i} f_j$, the learner has to contend with the fact that rewards corresponding to certain leaf nodes could have been chosen more than once by the adversary, giving reward $> \frac{1}{i}$. Thus, the learner still needs to visit all leaf nodes, which again takes $\Omega(|\mathcal{A}|^T)$ interactions with the environment

□

## A.3. Proof of Lemma 3.2

*Proof.* At the $t$th iteration of `MMDP`, we are solving a two-player zero-sum game over strategy spaces $\Pi$ and $\mathcal{F}_r$ with payoff given by Equation 1. All interaction with the environment happens during the collection of $\mathcal{D}_t$ so we analyze how many iterations $M$ we must perform to estimate the payoff matrix within $\epsilon_t$ uniformly w.p $\geq 1-\delta$.

First, note that there are $C = |\Pi||\mathcal{F}_r|$ elements in the matrix. Second, observe that each element of the matrix is within $[-T, T]$ before the $\frac{1}{T}$ normalization. Third, notice that the outer expectation in the first half of Equation 1 is taken with respect to the policy while we collect data by sampling $a_t$ uniformly at random. Thus, to estimate this term, we use

importance weighting between the learner and uniform policies. The maximum value of such a weight (corresponding to a deterministic learner policy) is $\frac{1}{|\mathcal{A}|}$. Thus, the overall scale of the random variable corresponding to each element of the payoff matrix is $R = T|\mathcal{A}|$. Now, applying Lemma A.1, we see that we need

$$M \geq O\left(\log\left(\frac{|\Pi||\mathcal{F}_r|}{\delta}\right)\frac{(T|\mathcal{A}|)^2}{\epsilon^2}\right) \tag{24}$$

trajectories, each of which could take $O(T)$ interactions with the environment, giving us an overall interaction complexity bound of

$$O\left(\log\left(\frac{|\Pi||\mathcal{F}_r|}{\delta}\right)\frac{T^3|\mathcal{A}|^2}{\epsilon^2}\right) \leq \text{poly}(T, |\mathcal{A}|, \frac{1}{\epsilon}, \log(\frac{1}{\delta}), \log(|\Pi|), \log(|\mathcal{F}_r|). \tag{25}$$

Observe that with this many samples, we are able to estimate all elements of the $\pi$ and $f$ payoff matrix to within $\epsilon$ w.p. $\geq 1-\delta$. Thus, the error we could accumulate by optimizing over the empirical rather than the population payoff matrix is bounded by $\epsilon \leq \epsilon_t$. □

### A.4. Proof of Theorem 3.3

*Proof.* Let $Q_f^{\pi^t \dots \pi^T}(s,a)$ denote the expected cumulative value of $f$ on trajectories generated by rolling out $\pi^t$ through $\pi^T$ starting from $(s,a)$. Then, via the Performance Difference Lemma (Kakade and Langford, 2002),

$$J(\pi_E) - J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{\xi\sim\pi_E}[Q_r^{\pi^{t+1}\dots\pi^T}(s_t,a_t) - \mathbb{E}_{a\sim\pi^t}[Q_r^{\pi^{t+1}\dots\pi^T}(s_t,a)]] \tag{26}$$

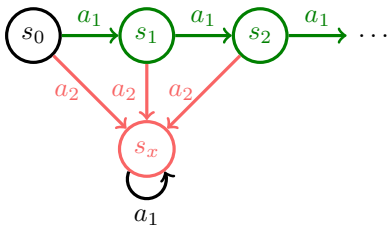
$$\leq \sum_{t=0}^{T} \sup_{f_t\in\mathcal{F}_r} \mathbb{E}_{\xi\sim\pi_E}[Q_{f_t}^{\pi^{t+1}\dots\pi^T}(s_t,a_t) - \mathbb{E}_{a\sim\pi^t}[Q_{f_t}^{\pi^{t+1}\dots\pi^T}(s_t,a)]] \tag{27}$$

$$\leq \sum_{t=0}^{T} T\epsilon_t = \bar{\epsilon}T^2. \tag{28}$$

□

### A.5. Proof of Theorem 3.4

*Proof.* We consider the CLIFF MDP of Swamy et al. (2021), which we reproduce here for convenience.




Assume the expert always takes $a_1$ and $r(s,a) = -\mathbb{1}_{s_x} - \mathbb{1}_{a_2}$. Thus, $J(\pi_E, r) = 0$. Assume that $\mathcal{F}_r = \{r\}$.

Let $\pi_a$ be the policy that takes $a_2$ with prob. $\epsilon T$ in $s_0$ and $a_1$ otherwise. Let $\pi_b$ be the policy that always takes $a_1$. Let $\pi = \{\pi_a, \pi_b, \dots\}$ be the sequence of policies returned by MMDP.

For the first $T-1$ steps of the algorithm, $\epsilon_t = 0$ as the learner plays $\pi_b$. On the last step of the algorithm, the learner picks a policy $\pi_a$ which makes mistakes for the rest of the horizon w.p. $\epsilon T$, giving it a moment matching error of $\epsilon_1 = \epsilon T$. Thus, overall, $\pi$ has average moment-matching error $\bar{\epsilon} = \frac{1}{T}(\epsilon T + \sum_{t=2}^{T} 0) = \epsilon$. However, on rollouts, the learner would have an $\epsilon T$ chance of paying a cost of 1 for the rest of the horizon, leading to a lower bound of $J(\pi_E, r) - J(\pi, r) = \epsilon T^2 \geq \Omega(\epsilon T^2)$. □

### A.6. Proof of Lemma 3.5

*Proof.* We proceed similarly to the proof of Theorem 3.2. All interaction with the environment happens during the $M$ interactions with the environment. As before, we are estimating a payoff matrix with $C = |\Pi||\mathcal{F}_r|$ elements within $\epsilon_i$ uniformly w.p $\geq 1 - \delta$. Each element has scale $R = T|\mathcal{A}|$. Applying Lemma A.1, we see that we need

$$M \geq O\left(\log\left(\frac{|\Pi||\mathcal{F}_r|}{\delta}\right)\frac{(T|\mathcal{A}|)^2}{\epsilon_i^2}\right) \tag{29}$$

trajectories, each of which could take $O(T)$ interactions with the environment, giving us an overall interaction complexity bound of

$$O\left(\log\left(\frac{|\Pi||\mathcal{F}_r|}{\delta}\right)\frac{T^3|\mathcal{A}|^2}{\epsilon^2}\right) \leq \text{poly}(T, |\mathcal{A}|, \frac{1}{\epsilon}, \log(\frac{1}{\delta}), \log(|\Pi|), \log(|\mathcal{F}_r|). \tag{30}$$

Observe that with this many samples, we are able to estimate all elements of the $\pi$ and $f$ payoff matrix to within $\epsilon$ w.p. $\geq 1 - \delta$. Thus, the error we could accumulate by optimizing over the empirical rather than the population payoff matrix is bounded by $\epsilon \leq \epsilon_i$. To complete the proof, observe that this bounds the optimization error (i.e. difference in value between $\pi_i$ and the per-round best response policy when plugged into Eq. 11) which upper bounds the instantaneous regret (i.e. difference in value between $\pi_i$ and the best-in-hindsight policy when plugged into Eq. 11).

□

### A.7. Proof of Theorem 3.6

*Proof.* First, we note that

$$J(\pi_E) - J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{\xi\sim\pi_E}[Q_r^\pi(s_t, a_t) - \mathbb{E}_{a\sim\pi}[Q_r^\pi(s_t, a)]] \tag{31}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{s,a\sim\rho_E^t}[\mathbb{E}_{\xi\sim\pi|s,a}[\sum_{\tau=t}^{T} r(s_\tau, a_\tau)]] - \mathbb{E}_{a'\sim\pi(s)}[\mathbb{E}_{\xi\sim\pi|s,a'}[\sum_{\tau=t}^{T} r(s_\tau, a_\tau)]]] \tag{32}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{s_t,a_t\sim\rho_t^\pi}[r(s_t, a_t)] - \mathbb{E}_{s_t,a_t\sim\rho_t^E}[r(s_t, a_t)]. \tag{33}$$

The first equality is via the PDL, the second via the definition of a $Q$ function, and the third by the definition of $J$. Next, we set

$$f_i = \arg\max_{f\in\mathcal{F}_r} \sum_{t=1}^{T} \mathbb{E}_{s_t,a_t\sim\rho_t^\pi}[f(s_t, a_t)] - \mathbb{E}_{s_t,a_t\sim\rho_t^E}[f(s_t, a_t)] \tag{34}$$

and define

$$L_i(\pi) = \frac{1}{T}\sum_{t=0}^{T} \mathbb{E}_{s\sim\rho_t^E}[\mathbb{E}_{a\sim\pi(s)}[\mathbb{E}_{\xi\sim\pi_i|s,a}[\sum_{\tau=t}^{T} f_i(s_\tau, a_\tau)]]]. \tag{35}$$

Note the iteration-indexed "roll-out" policy. We use this sequence of loss functions to define a regret measure,

$$\bar{\epsilon} = \frac{1}{NT}\sum_{i=1}^{N} L_i(\pi_i) - \min_{\pi\in\Pi}\frac{1}{NT}\sum_{i=1}^{N} L_i(\pi) \in [-1, 1], \tag{36}$$

and $\bar{\pi}$ to denote the uniform mixture over policy iterates. Now, by our earlier equalities,

$$J(\pi_E) - J(\bar{\pi}) = \frac{1}{N}\sum_{i=1}^{N} J(\pi_E) - J(\pi_i) \tag{37}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \mathbb{E}_{s,a\sim\rho_E^t}[\mathbb{E}_{\xi\sim\pi_i|s,a}[\sum_{\tau=t}^{T} r(s_\tau,a_\tau)]] - \mathbb{E}_{a'\sim\pi_i(s)}[\mathbb{E}_{\xi\sim\pi_i|s,a'}[\sum_{\tau=t}^{T} r(s_\tau,a_\tau)]]] \tag{38}$$

$$\leq \sup_{f\in\mathcal{F}_r} \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \mathbb{E}_{s,a\sim\rho_E^t}[\mathbb{E}_{\xi\sim\pi_i|s,a}[\sum_{\tau=t}^{T} f(s_\tau,a_\tau)]] - \mathbb{E}_{a'\sim\pi_i(s)}[\mathbb{E}_{\xi\sim\pi_i|s,a'}[\sum_{\tau=t}^{T} f(s_\tau,a_\tau)]]] \tag{39}$$

$$\leq \frac{1}{N}\sum_{i=1}^{N} \sup_{f_i\in\mathcal{F}_r} \sum_{t=1}^{T} \mathbb{E}_{s,a\sim\rho_E^t}[\mathbb{E}_{\xi\sim\pi_i|s,a}[\sum_{\tau=t}^{T} f_i(s_\tau,a_\tau)]] - \mathbb{E}_{a'\sim\pi_i(s)}[\mathbb{E}_{\xi\sim\pi_i|s,a'}[\sum_{\tau=t}^{T} f_i(s_\tau,a_\tau)]]] \tag{40}$$

$$= \frac{1}{N}\sum_{i=1}^{N} T(L_i(\pi_i) - L_i(\pi_E)). \tag{41}$$

Set $\pi^* = \arg\min_{\pi\in\Pi}\sum_{i=1}^{N} L_i(\pi)$. Continuing,

$$J(\pi_E) - J(\bar{\pi}) \leq \frac{1}{N}\sum_{i=1}^{N} T(L_i(\pi_i) - L(\pi^*)) \tag{42}$$

$$= \bar{\epsilon}T^2. \tag{43}$$

Then, because at least one member in a sequence must perform as well as the mixture, we know that $J(\pi_E) - J(\pi) \leq \bar{\epsilon}T^2$, where $\pi \in \{\pi_1, \ldots, \pi_N\}$ is the member with the lowest validation error. □

### A.8. Proof of Theorem 3.7

*Proof.* We define $L_i$ and $\bar{\epsilon}$ as before, i.e.

$$L_i(\pi, f) = \frac{1}{T}\sum_{t=0}^{T} \mathbb{E}_{s\sim\rho_t^E}[\mathbb{E}_{a\sim\pi(s)}[\mathbb{E}_{\xi\sim\pi_i|s,a}[\sum_{\tau=t}^{T} f(s_\tau,a_\tau)]]], \tag{44}$$

$$\bar{\epsilon} = \frac{1}{NT}\sum_{i=1}^{N} L_i(\pi_i, f_i) - \min_{\pi\in\Pi}\frac{1}{NT}\sum_{i=1}^{N} L_i(\pi, f_i) \in [-1, 1]. \tag{45}$$

Additionally, we define an average regret measure for the adversary:

$$\bar{\delta} = \max_{f\in\mathcal{F}_r}\frac{1}{NT}\sum_{i=1}^{N} L_i(\pi_i, f) - \frac{1}{NT}\sum_{i=1}^{N} L_i(\pi_i, f_i) \in [-1, 1]. \tag{46}$$

Note that

$$\bar{\epsilon} + \bar{\delta} = \frac{1}{NT}(\max_{f\in\mathcal{F}_r}\sum_{i=1}^{N} L_i(\pi_i, f) - \min_{\pi\in\Pi}\sum_{i=1}^{N} L_i(\pi, f_i)). \tag{47}$$

Proceeding as before,

$$J(\pi_E) - J(\bar{\pi}) \leq \max_{f\in\mathcal{F}_r}\frac{1}{N}\sum_{i=1}^{N} T(L_i(\pi_i, f) - L(\pi^*, f)) \tag{48}$$

$$= \frac{1}{N}\sum_{i=1}^{N} T(\bar{\epsilon}T + \bar{\delta}T) \tag{49}$$

$$= (\bar{\epsilon} + \bar{\delta})T^2. \tag{50}$$

□

### A.9. Proof of Theorem 3.8

*Proof.* We again consider the CLIFF MDP. As before, assume that the expert always takes $a_1$, $r(s,a) = -\mathbb{1}_{s_x} - \mathbb{1}_{a_2}$, and that $\mathcal{F}_r = \{r\}$.

Let $\pi$ be a policy that takes $a_2$ in $s_0$ with prob. $\epsilon T$ and $a_1$ with prob. 1 everywhere else. Thus, on a $\frac{T-1}{T}$ fraction of the rollouts, there is no difference between the learner on the expert. On the $\frac{1}{T}$ fraction of rollouts that start from $s_0$, the learner diverges from the expert for the entire horizon with probability $\epsilon T$, so the discriminator can penalize it $\epsilon T^2$ on average. Putting it all together, $\bar{\epsilon} = \frac{1}{T}[\frac{T-1}{T}(0) + \frac{1}{T}(\epsilon T)] = \epsilon$. The outer $\frac{1}{T}$ comes from the average over timesteps in the payoff.

On rollouts, the learner would have an $\epsilon T$ chance of paying a cost of 1 for the rest of the horizon (as they always start at $s_0$), leading to a lower bound of $J(\pi_E, r) - J(\pi, r) = \epsilon T^2 \geq \Omega(\epsilon T^2)$. □

We note that if we started on the true start-state distribution ($\alpha = 0$), we would instead get an $\bar{\epsilon} = \epsilon T$ and therefore a bound linear in the horizon, recovering the lower bound results in Swamy et al. (2021).

### A.10. Proof of Theorem 3.9

*Proof.* Consider the following MDP with one of two rewards depicted in Figure 3.

*Example* 1 (**Forked Tree MDP**). The learner always starts at the top of the tree and continues until they reach the bottom (i.e. $T = 2$). They are allowed to choose between three policies: $\pi_E$, $\pi_1$, and $\pi_2$, which correspond to always going left, center, or right, respectively. The adversary chooses between true reward function $r$ and $\tilde{r}$. The learner always starts at $\pi_1$ and the adversary at $\tilde{r}$. □

Let $J_E^k(\pi, f) = \frac{1}{T}\sum_t^T \mathbb{E}_{s\sim\rho_E^t}[f(s,\pi(s)) + V^{\pi_k,f}(s')]$, where $V^{\pi_k,f}(s')$ is the value function of $\pi_k$ under $f$ evaluated at successor state $s'$. We now compute the payoff matrices the different algorithms will utilize.

$J(\pi,f) - J(\pi_E,f)$

| | $r$ | $\tilde{r}$ |
|---|---|---|
| $\pi_E$ | 0 | 0 |
| $\pi_1$ | $-2$ | $-3$ |
| $\pi_2$ | $-2$ | $-3$ |

$J_E^1(\pi,f)$

| | $r$ | $\tilde{r}$ |
|---|---|---|
| $\pi_E$ | 1 | 1.5 |
| $\pi_1$ | 0 | 0 |
| $\pi_2$ | 0 | 2 |

$J_E^2(\pi,f)$

| | $r$ | $\tilde{r}$ |
|---|---|---|
| $\pi_E$ | 1 | 1.5 |
| $\pi_1$ | 0 | 2 |
| $\pi_2$ | 0 | 0 |

$J_E^E(\pi,f)$

| | $r$ | $\tilde{r}$ |
|---|---|---|
| $\pi_E$ | 1 | 2 |
| $\pi_1$ | 0 | 0 |
| $\pi_2$ | 0 | 0 |

We first compare the four algorithms that produce stationary policies: `NRMM` (both best-response and no-regret variants), `DUAL` (no-regret discriminator against a best response via NRPI), and `MM` (Algorithms 1 / 2). The discriminator always plays a no-regret or best response strategy using the first payoff matrix. `MM` also uses the first for policy search. `NRMM(BR)` and `NRMM(NR)` both use the other three payoff matrices for policy search in a no-regret fashion – the difference between these two approaches is whether the discriminator follows a no-regret or best-response strategy. `DUAL` also uses the other three payoff matrices for policy search but does so in a best-response fashion (i.e only using the matrix that corresponds to rollouts with the previous policy).

`FILTER(BR)`

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $\pi_1$ | $\tilde{r}$ |
| 2 | $\pi_2$ | $\tilde{r}$ |
| 3 | $\pi_E$ | $r/\tilde{r}$ |
| 4 | $\downarrow$ | $\downarrow$ |

✓

`FILTER(NR)`

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $\pi_1$ | $\tilde{r}$ |
| 2 | $\pi_2$ | $\tilde{r}$ |
| 3 | $\pi_E$ | $\tilde{r}$ |
| 4 | $\downarrow$ | $\downarrow$ |

✓

`DUAL`

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $\pi_1$ | $\tilde{r}$ |
| 2 | $\pi_2$ | $\tilde{r}$ |
| 3 | $\pi_1$ | $\tilde{r}$ |
| 4 | $\pi_2$ | $\tilde{r}$ |

✗

Algorithms 1 / 2

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $\pi_1$ | $\tilde{r}$ |
| 2 | $\pi_E$ | $r/\tilde{r}$ |
| 3 | $\downarrow$ | $\downarrow$ |
| 4 | ... | ... |

✓

We see that all algorithms that produce stationary policies other than `DUAL` eventually converge to the expert policy. We next consider `MMDP` and compute the relevant quantities:

$$J([\pi_a, \pi_b], f) - J([\pi_E, \pi_E], f)$$

| | $r$ | $\tilde{r}$ |
|---|---|---|
| $[\pi_E, \pi_E]$ | 0 | 0 |
| $[\pi_E, \pi_1]$ | 0 | 0 |
| $[\pi_E, \pi_2]$ | 0 | 0 |
| $[\pi_1, \pi_E]$ | $-2$ | $-3$ |
| $[\pi_1, \pi_1]$ | $-2$ | $-3$ |
| $[\pi_1, \pi_2]$ | $-2$ | 1 |
| $[\pi_2, \pi_E]$ | $-2$ | $-3$ |
| $[\pi_2, \pi_1]$ | $-2$ | 1 |
| $[\pi_2, \pi_2]$ | $-2$ | $-3$ |

We first consider $t = T = 2$, where the first policy is fixed to be $\pi_E$. Notice that no matter what policy the learner picks, they receive the same payoff under any strategy the adversary chooses. We therefore consider all three cases for the next game ($t = 1$). For simplicity, we assume both players follow a no-regret strategy.

`MMDP` ($\pi_E$ Suffix)

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $[\pi_1, \pi_E]$ | $\tilde{r}$ |
| 2 | $[\pi_E, \pi_E]$ | $\tilde{r}/r$ |
| 3 | $\downarrow$ | $\downarrow$ |

`MMDP` ($\pi_1$ Suffix)

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $[\pi_1, \pi_1]$ | $\tilde{r}$ |
| 2 | $[\pi_2, \pi_1]$ | $r$ |
| 3 | $[\pi_E, \pi_1]$ | $r$ |
| 4 | $\downarrow$ | $\downarrow$ |

`MMDP` ($\pi_2$ Suffix)

| # | $\pi$ | $f$ |
|---|---|---|
| 1 | $[\pi_1, \pi_2]$ | $r$ |
| 2 | $[\pi_E, \pi_2]$ | $\tilde{r}/r$ |
| 3 | $\downarrow$ | $\downarrow$ |

In all three cases, `MMDP` converges to a policy sequence that is value equivalent to the expert under all reward functions. □

**Theorem A.2.** *The trajectory-based sampling procedure implied by Equation 33 is lower variance than the suffix-based sampling procedure implied by Equation 32.*

### A.11. Proof of Theorem A.2

*Proof.* First, let us explicitly define the sampling procedure implied by each of the above. At step $t$:

- Equation (32): Sample a state-action pair from the expert visitation distribution at timestep $t$. Reset the learner to this state. Execute the sampled action and then roll out the learner for $T - t$ timesteps, adding up the reward function along this suffix. Sample a state-action pair from the expert visitation distribution at timestep $t$. Reset the learner to this state. Roll out the learner for $T - t + 1$ timesteps, adding up the reward function along this suffix. Record the difference of these two suffix sums.
- Equation (33): Roll out the current learner policy for $t$ timesteps. Use the sample at timestep $t$ for evaluating the reward function. Sample a state-action pair from the expert visitation distribution at timestep $t$. Use this sample for evaluating the reward function. Record the difference of these two single-step evaluations.

We consider two settings:

1. Total independence between timesteps: $\forall t \in [T]$, $\text{Var}(r(s_t, a_t)) = \sigma^2$.
2. Total dependence (determinism) between timesteps: $\text{Var}(r(s_0, a_0)) = \sigma^2, \forall t \in [1, T], r(s_t, a_t) = r(s_0, a_0)$.

Let's begin with Case 1. Equation (32) has variance

$$\sum_t^T (T - t)\sigma^2 + (T - t)\sigma^2 = T(T - 1)\sigma^2, \tag{51}$$

while Equation (33) has variance

$$\sum_{t}^{T} \sigma^2 + \sigma^2 = 2T\sigma^2. \tag{52}$$

Observe that $2T\sigma^2 < T(T-1)\sigma^2$ to complete this case. Similarly, for Case 2, Equation (32) has variance

$$\sum_{t}^{T} (T-t)^2\sigma^2 + (T-t)^2\sigma^2 = \frac{\sigma^2(T)(T+1)(2T+1)}{3}, \tag{53}$$

and Equation (33) has variance

$$\sum_{t}^{T} \sigma^2 + \sigma^2 = 2T\sigma^2. \tag{54}$$

The former variance is again greater than the latter, completing this case and the proof. □

## B. Experiments

We use Optimistic Adam (Daskalakis et al., 2017) for all policy and discriminator optimization, taking advantage of its speed and last-iterate convergence properties. We use gradient penalties (Gulrajani et al., 2017) to stabilize our discriminator training for all algorithms. Our policies, value functions, and discriminators are all 2-layer ReLu networks with a hidden size of 256. Each outer loop iteration lasts for 5000 steps of environment interaction. We sample 4 trajectories to use in the discriminator update at the end of each outer-loop iteration.

### B.1. PyBullet Tasks

For the PyBullet tasks (Walker, Hopper, HalfCheetah), we use the Soft Actor Critic (Haarnoja et al., 2018) implementation provided by Raffin et al. (2019) for policy optimization for both the expert and the learner. We use the hyperparameters in Table 4 for all experiments. We train behavioral cloning for 100,000 steps.

| PARAMETER | VALUE |
|---|---|
| BUFFER SIZE | 300000 |
| BATCH SIZE | 256 |
| $\gamma$ | 0.98 |
| $\tau$ | 0.02 |
| TRAINING FREQ. | 64 |
| GRADIENT STEPS | 64 |
| LEARNING RATE | LIN. SCHED. 7.3E-4 |
| POLICY ARCHITECTURE | 256 X 2 |
| STATE-DEPENDENT EXPLORATION | TRUE |
| TRAINING TIMESTEPS | 1E6 |

Table 4: Expert and learner hyperparameters for SAC.

We use $\alpha = 0.5$ for both variants of `FILTER` as we found it to perform better than $\alpha = 1$.

For our discriminator, we start with a learning rate of $8e-3$ and decay it linearly over outer-loop iterations.

### B.2. D4RL Tasks

For the D4RL tasks (both large `antmazes`), we use the data provided by Fu et al. (2020) as our expert demonstrations. We give all algorithms access to goal information by appending it to the observation. This helps explain why our behavioral cloning baseline significantly out-performs previously published results and might be of independent interest to the Offline RL community. [6] Importantly, we did not filter the data down whatsoever as in the "%-BC" approach of Chen et al. (2021), so our algorithms are all truly reward-free.

For our policy optimizer, we build upon the TD3+BC implementation of Fujimoto and Gu (2021) with the default hyperparameters. For behavioral cloning, we run the optimizer for 500k steps while zeroing out the component of the actor update that depends on rewards.

For `MM` and `FILTER`, we pre-train the policy with 10,000 steps of behavioral cloning. We use a dual replay buffer strategy, similar to that of Hester et al. (2018); Reddy et al. (2019); Swamy et al. (2021). One buffer contains expert demonstrations while the other contains learner rollouts. We sample a batch from one with equal probability for each policy update. For samples from the expert buffer, we use the current discriminator to impute rewards and use the BC regularizer term. For samples from the learner's buffer, we use the recorded discriminator values and turn off the BC regularizer. We use $\alpha = 1$ for `FILTER` (i.e. `NRMM`).

For our discriminator, we start with a learning rate of $8e-4$ and decay it linearly over outer-loop iterations.

[6] We found that on the small and medium mazes, a properly tuned implementation of BC was able to achieve scores upwards of 70.