Model-Free Imitation Learning with Policy Optimization

Jonathan Ho HOJ@CS.STANFORD.EDU
Jayesh K. Gupta JKG@CS.STANFORD.EDU
Stefano Ermon ERMON@CS.STANFORD.EDU
Stanford University

Abstract

In imitation learning, an agent learns how to behave in an environment with an unknown cost function by mimicking expert demonstrations. Existing imitation learning algorithms typically involve solving a sequence of planning or reinforcement learning problems. Such algorithms are therefore not directly applicable to large, high-dimensional environments, and their performance can significantly degrade if the planning problems are not solved to optimality. Under the apprenticeship learning formalism, we develop alternative model-free algorithms for finding a parameterized stochastic policy that performs at least as well as an expert policy on an unknown cost function, based on sample trajectories from the expert. Our approach, based on policy gradients, scales to large continuous environments with guaranteed convergence to local minima.

1. Introduction

To use reinforcement learning, the learner needs access to a cost or reward signal to identify desirable outcomes. The dependence between the cost function and the corresponding optimal policy, however, is generally complex, as it involves planning. In practice, eliciting a cost function that achieves desired behavior can be difficult (Bagnell, 2015). An alternative and often more practical approach, called *imitation learning*, is to encode preferences and differentiate between desirable and undesirable outcomes using demonstrations provided by an expert (Pomerleau, 1991; Russell, 1998).

The simplest approach to imitation learning is behavioral cloning, in which the goal is to learn the relationship between states and optimal actions as a supervised learning problem (Pomerleau, 1991). While conceptually simple and theoretically sound (Syed & Schapire, 2010), small inaccuracies of the learned model compound over time, and can lead to situations that are quite different from the ones encountered during training. This is often referred to as the problem of *cascading errors*, and is related to *covariate shift* (Ross & Bagnell, 2010; Bagnell, 2015).

Inverse reinforcement learning (IRL) methods (Russell, 1998; Ng & Russell, 2000; Ratliff et al., 2006; Ziebart et al., 2008), which are some of the most successful approaches to imitation learning, assume that the behavior the learner desires to imitate is generated by an expert behaving optimally with respect to an unknown cost function. IRL algorithms train models over entire trajectories of behavior instead of individual actions, and hence do not suffer from cascading error problems. Furthermore, because the assumption of expert optimality acts as a prior on the space of policies, IRL algorithms can allow the learner to generalize expert behavior to unseen states much more effectively than if the learner had tried to produce a policy or value function instead (Ng & Russell, 2000; Bagnell, 2015).

Unfortunately, the assumption of expert optimality leads to expensive design choices in IRL algorithms. At each iteration, to determine whether a certain cost function c fits an expert policy π_E, the IRL algorithm must compare the return of π_E with the return of all other possible policies. Most IRL algorithms do this by running a reinforcement learning algorithm on c (Neu & Szepesvári, 2009). Because reinforcement learning must be run at each iteration, IRL algorithms can be extremely expensive to run in large domains.

We forgo learning a cost function. We propose a method that directly learns a policy from expert trajectories, exploiting for learning signal a *class* of cost functions, which distinguish the expert policy from all others. We first develop a simple, unified view of a certain class of imitation learning algorithms called *apprenticeship learning* algorithms (Abbeel & Ng, 2004; Syed et al., 2008). This view naturally leads to the development of a gradient-based optimization formulation over parameterized policies for ap-

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

prenticeship learning. We then provide two model-free realizations of these optimization algorithms: one is based on a standard policy gradient algorithm, and the other is based on a recently developed policy gradient algorithm that incorporates trust region constraints to stabilize optimization. We demonstrate the effectiveness of our approach on control problems with very high-dimensional observations (over 600 continuous features), for which we train neural network control policies from scratch.

2. Preliminaries

We begin by defining basic notions from reinforcement learning. We are given an environment consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a dynamics model $p(s'|s,a)$, and an initial state distribution $p_0(s_0)$. Agents act according to stationary stochastic policies $\pi(a|s)$, which specify action choice probabilities for each state. We will work with finite $\mathcal{S}$ and $\mathcal{A}$, but our methods will extend to continuous spaces.

With respect to a cost function $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor $\gamma \in [0,1)$, and a policy π, the *state-action value* function $Q^c_\pi(s,a)$, the *state value* function $V^c_\pi(s)$, and the *advantage* function $A^c_\pi(s,a)$ are defined as $Q^c_\pi(s_t,a_t) = \mathbb{E}_{p_0,p,\pi}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} c(s_{t'},a_{t'})\right]$, $V^c_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^c_\pi(s,a)\right]$, and $A^c_\pi(s,a) = Q^c_\pi(s,a) - V^c_\pi(s)$. The *expected cost* of π is $\eta^c(\pi) = \mathbb{E}_{s_0\sim p_0}\left[V^c_\pi(s_0)\right]$. For clarity, when c is a parameterized function, written as c_w for a parameter vector w, we will replace c by w in the names of these quantities—for example, η^w, Q^w_π, etc. We define the γ-discounted *state visitation distribution* of a policy π by $\rho_\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_{p_0,\pi}\left[s_t = s\right]$, where $\mathbb{P}_{p_0,\pi}\left[s_t = s\right]$ is the probability of landing in state s at time t, when following π starting from $s_0 \sim p_0$. When convenient, we will overload notation for *state-action visitation distributions*: $\rho_\pi(s,a) = \pi(a|s)\rho_\pi(s)$, allowing us to write expected cost as $\eta^c(\pi) = \sum_{s,a} \rho_\pi(s,a)\, c(s,a) = \mathbb{E}_{\rho_\pi}[c(s,a)]$.

3. Apprenticeship learning

To address the imitation learning problem, we adopt the apprenticeship learning formalism, in which the learner must find a policy that performs at least as well as the expert π_E on an unknown true cost function $c_{\text{true}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (Abbeel & Ng, 2004; Syed et al., 2008). Formally, the learner's goal is to find a policy π such that $\eta^{c_{\text{true}}}(\pi) \le \eta^{c_{\text{true}}}(\pi_E)$, given a dataset of trajectory samples from π_E.

To do this, apprenticeship learning algorithms carry with them the assumption that the true cost function belongs to a class of cost functions $\mathcal{C}$. Accordingly, they seek a policy π that performs as well as π_E for all $c \in \mathcal{C}$—that is, they seek to satisfy the constraints

$$\eta^c(\pi) \le \eta^c(\pi_E) \quad \text{for all} \quad c \in \mathcal{C} \tag{1}$$

Because $c_{\text{true}} \in \mathcal{C}$ by assumption, satisfying this family of constraints ensures successful apprenticeship learning. We can reformulate this constraint satisfaction problem as an optimization problem by defining the objective

$$\delta_{\mathcal{C}}(\pi, \pi_E) = \sup_{c\in\mathcal{C}} \eta^c(\pi) - \eta^c(\pi_E) \tag{2}$$

Intuitively, the cost functions in $\mathcal{C}$ distinguish the expert from all other policies, assigning high expected cost to non-expert policies and low expected cost to the expert policy. If $\delta_c(\pi,\pi_E) > 0$, then there exists some cost in $\mathcal{C}$ such that π performs worse than π_E—in this case, π is a poor solution to the apprenticeship learning problem. On the other hand, if $\delta_c(\pi,\pi_E) \le 0$, then π performs at least as well as π_E for all costs in $\mathcal{C}$, and therefore satisfies the apprenticeship learning constraints (1).

Having defined the objective, the job of an apprenticeship learning algorithm is to solve the optimization problem

$$\underset{\pi}{\text{minimize}}\ \delta_{\mathcal{C}}(\pi,\pi_E). \tag{3}$$

So far, we have described a general framework for defining apprenticeship learning algorithms. To instantiate this framework, two ingredients must be provided: a cost function class $\mathcal{C}$, and an optimization algorithm to solve (3). Our goal in this paper is to address the optimization ingredient, so we will use linearly parameterized cost functions in our experiments, although the development of our method is agnostic to the particulars of the cost function class. In Section 4, we will develop a method for approximately solving (3) over a class of parameterized stochastic policies (for example, neural network policies), assuming generic access to a method for solving the maximization (2) over costs for fixed policies π. Our method will perform gradient-based stochastic optimization on policy parameters—we refer to this strategy as *policy optimization*.

3.1. Examples from prior work

Before delving into our method, we first review two prototypical examples of apprenticeship learning algorithms. We show how they fall into the framework detailed in this section (namely, how they choose the cost function class $\mathcal{C}$), and we briefly describe their solution techniques for solving (3), which differ vastly from our new policy optimization method.

Feature expectation matching Abbeel & Ng (2004) define $\mathcal{C}$ by first fixing a set of basis cost functions $c_1, \ldots, c_k$,

where $c_j : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and then defining the cost class as a certain set of linear combinations of these basis functions:

$$\mathcal{C}_{\text{linear}} = \left\{ c_w \triangleq \textstyle\sum_{i=1}^k w_i c_i \;\middle|\; \|w\|_2 \le 1 \right\} \tag{4}$$

The structure of $\mathcal{C}_{\text{linear}}$ allows the expected costs with respect to c_w to be written as an inner product of w with a certain *feature expectation* vector of π, defined as $\phi(\pi) \triangleq \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^\infty \gamma^t \phi(s_t, a_t)\right]$, where $\phi(s,a) = [c_1(s,a) \cdots c_k(s,a)]^T$. Because any cost in $\mathcal{C}_{\text{linear}}$ can be written as $c_w(s,a) = w \cdot \phi(s,a)$, linearity of expectation yields $\eta^w(\pi) = \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t w \cdot \phi(s_t, a_t)\right] = w \cdot \phi(\pi)$.

Based on this observation, Abbeel & Ng (2004) propose to match feature expectations; that is, to find a policy π such that $\phi(\pi) \approx \phi(\pi_E)$, thereby guaranteeing that $\eta^w(\pi) = w \cdot \phi(\pi) \approx w \cdot \phi(\pi_E) = \eta^w(\pi_E)$ for all cost functions $c_w \in \mathcal{C}_{\text{linear}}$. We can understand feature expectation matching as minimization of $\delta_{\mathcal{C}_{\text{linear}}}$, because

$$\begin{aligned}
\delta_{\mathcal{C}_{\text{linear}}}(\pi, \pi_E) &= \sup_{\|w\|_2 \le 1} \mathbb{E}_{\rho_\pi}[w \cdot \phi(s,a)] - \mathbb{E}_{\rho_{\pi_E}}[w \cdot \phi(s,a)] \\
&= \sup_{\|w\|_2 \le 1} w \cdot (\phi(\pi) - \phi(\pi_E)) \\
&= \|\phi(\pi) - \phi(\pi_E)\|_2.
\end{aligned} \tag{5}$$

To solve this problem, Abbeel & Ng (2004) propose to incrementally generate a set of policies by inverse reinforcement learning (Ng & Russell, 2000). At each iteration, their algorithm finds a cost function that assigns low expected cost to the expert and high expected cost to previously found policies. Then, it adds to the set of policies the optimal policy for this cost, computed via reinforcement learning. These steps are repeated until there is no cost function that makes the expert perform much better than the previously found policies. The final policy, which minimizes (5), is produced by stochastically mixing the generated policies using weights calculated by a quadratic program. This algorithm is quite expensive to run for large MDPs, because it requires running reinforcement learning at each iteration.

Game-theoretic approaches Syed & Schapire (2007); Syed et al. (2008) proposed two apprenticeship learning algorithms, Multiplicative Weights Apprenticeship Learning (MWAL) and Linear Programming Apprenticeship Learning (LPAL), that also use basis cost functions, but with the weights restricted to give a convex combination:

$$\mathcal{C}_{\text{convex}} = \left\{ c_w \triangleq \textstyle\sum_{i=1}^k w_i c_i \;\middle|\; w_i \ge 0,\ \sum_i w_i = 1 \right\} \tag{6}$$

MWAL uses a multiplicative weights update method to solve the resulting optimization problem, and like Abbeel and Ng's method, requires running reinforcement learning in its inner loop. Syed et al. (2008) address this computational complexity with their LPAL method. They notice that restricting the weights on the basis functions to lie on the simplex allows the maximization over costs to be performed instead over a finite set: the problem (3) with $\mathcal{C}_{\text{convex}}$ can be written as $\min_\pi \max_{i \in \{1,\ldots,k\}} \eta^{c_i}(\pi) - \eta^{c_i}(\pi_E)$. Syed et al. are therefore able to formulate a single linear program on state-action visitation frequencies that simultaneously encodes (3) and Bellman flow constraints on the frequencies to ensure that they can be generated by some policy in the environment.

Inspired by LPAL, we will formulate an unconstrained optimization approach to apprenticeship learning. We propose to optimize (3) directly over parameterized stochastic policies instead of state-action visitation distributions, allowing us to scale to large spaces without keeping variables for each state and action. We keep our formulation general enough to allow for any cost function class $\mathcal{C}$, but because the focus of this paper is the optimization over policies, we will use linearly parameterized cost classes $\mathcal{C}_{\text{linear}}$ and $\mathcal{C}_{\text{convex}}$ in our experiments. (Note that despite assuming linearity, this setting is already more general than LPAL, as the maximization over $\mathcal{C}_{\text{linear}}$ cannot be written as a maximization over a finite set.)

4. Policy optimization for apprenticeship learning

Having reviewed existing algorithms for solving various settings of apprenticeship learning, we now delve into policy optimization strategies that directly operate on (3) as a stochastic optimization problem. To allow us to scale to large, continuous environments, we first fix a class of smoothly parameterized stochastic policies $\Pi = \{\pi_\theta \mid \theta \in \Theta\}$, where Θ is a set of valid parameter vectors. With this class of policies, our goal is to solve the optimization problem (3) over policy parameters:

$$\underset{\theta}{\text{minimize}}\ \delta_{\mathcal{C}}(\pi_\theta, \pi_E)$$

where $\delta_{\mathcal{C}}(\pi_\theta, \pi_E) = \sup_{c \in \mathcal{C}} \eta^c(\pi_\theta) - \eta^c(\pi_E)$. We propose find a local minimum to this problem using gradient-based stochastic optimization. To this end, let us first examine the gradient of $\delta_{\mathcal{C}}$ with respect to θ. Letting c^* denote the cost function that achieves the supremum in δ_C,[1] we have

$$\nabla_\theta \delta_{\mathcal{C}}(\pi_\theta, \pi_E) = \nabla_\theta \eta^{c^*}(\pi_\theta) \tag{7}$$

This formula dictates the basic structure a gradient-based algorithm must take to minimize $\delta_{\mathcal{C}}$—it must first compute c^* for a fixed θ, then it must use this c^* to improve θ for the next iteration. (Our algorithm in Section 4.2 will actually have to identify a cost function defined by a more complicated criterion, but as we will see in Section 4.3,

[1] In this paper, we only work with classes $\mathcal{C}$ for which this supremum is achieved.

Algorithm 1 IM-REINFORCE

Input: Expert trajectories τ_E, initial policy parameters. θ_0
for $i = 0, 1, 2, \ldots$ **do**
  Roll out trajectories $\tau \sim \pi_{\theta_i}$
  Compute $\hat{c}$ achieving the supremum in (10), with expectations taken over τ and τ_E
  Estimate the gradient $\nabla_\theta \eta^{\hat{c}}(\pi_\theta)|_{\theta=\theta_i}$ (8) with τ
  Use the gradient to take a step from θ_i to θ_{i+1}
end for

this will not pose significant difficulty.) Because the cost c^* effectively defines a reinforcement learning problem at the current policy π_θ, we can interpret gradient-based optimization of $\delta_{\mathcal{C}}$ as a procedure that alternates between (1) fitting a local reinforcement learning problem to generate learning signal for imitation, and (2) improving the policy with respect to this local problem. We will discuss how to find c^* in Section 4.3; for now, we will only discuss strategies for policy improvement.

4.1. Policy gradient

The most straightforward method of optimizing (3) is stochastic gradient descent with an estimate of the gradient (7):

$$\nabla_\theta \eta^{c^*}(\pi_\theta) = \mathbb{E}_{\rho_{\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{c^*}_{\pi_\theta}(s,a)\right] \quad (8)$$

This is the classic policy gradient formula for the cost function c^* (Sutton et al., 1999). To estimate this from samples, we propose the following algorithm, called IM-REINFORCE, which parallels the development of REINFORCE (Williams, 1992) for reinforcement learning. As input, IM-REINFORCE is given expert trajectories (that is, rollouts of the expert policy). At each iteration, for the current parameter vector θ_0, trajectories are sampled using $\pi_0 \triangleq \pi_{\theta_0}$. Then, the cost $\hat{c}$ attaining the supremum of an empirical estimate of $\delta_{\mathcal{C}}$ is calculated to satisfy:

$$\hat{\delta}_{\mathcal{C}}(\pi_0, \pi_E) = \sup_{c \in \mathcal{C}} \hat{\mathbb{E}}_{\rho_{\pi_0}}[c(s,a)] - \hat{\mathbb{E}}_{\rho_{\pi_E}}[c(s,a)] \quad (9)$$

$$= \hat{\mathbb{E}}_{\rho_{\pi_0}}[\hat{c}(s,a)] - \hat{\mathbb{E}}_{\rho_{\pi_E}}[\hat{c}(s,a)] \quad (10)$$

Here, $\hat{\mathbb{E}}$ denotes empirical expectation using rollout samples. We describe how to compute $\hat{c}$ in detail in Section 4.3; how to do so depends on $\mathcal{C}$.

With $\hat{c}$, IM-REINFORCE then estimates the gradient $\nabla_\theta \eta^{\hat{c}}$ using the formula (8), where the state-action value $Q^{\hat{c}}_{\pi_0}(s,a)$ is estimated using discounted future sums of $\hat{c}$ costs along rollouts for π_0. Finally, to complete the iteration, IM-REINFORCE takes a step in the resulting gradient direction, producing new policy parameters θ ready for the next iteration. These steps are summarized in Algorithm 1.

4.2. Monotonic policy improvements

While IM-REINFORCE is straightforward to implement, the gradient estimator (8) exhibits extremely high variance, making the algorithm very slow to converge, or even diverge for reasonably large step sizes. This variance issue is not unique to our apprenticeship learning formulation, and is a hallmark difficulty of policy gradient algorithms for reinforcement learning (Peters & Schaal, 2008).

The reinforcement learning literature contains a vast number of techniques for calculating high-quality policy parameter steps based on Monte Carlo estimates of the gradient. We make no attempt to fully review these techniques here. Instead, we will directly draw inspiration from a recently developed algorithm called *trust region policy optimization* (TRPO), a model-free policy search algorithm capable of quickly training large neural network stochastic policies for complex tasks (Schulman et al., 2015).

TRPO for reinforcement learning In this section, we will review TRPO for reinforcement learning, and in the next, we will develop an analogous algorithm for our apprenticeship learning setting. For now, we will drop the cost function superscript c, because the cost is fixed in the reinforcement learning setting.

Suppose we have a current policy π_0 that we wish to improve. We can write the performance of a new policy π in terms of the performance of π_0 (Kakade & Langford, 2002):

$$\eta(\pi) = \eta(\pi_0) + \mathbb{E}_{\rho_\pi}\mathbb{E}_{a\sim\pi(\cdot|s)}[A_{\pi_0}(s,a)] \quad (11)$$

Vanilla policy gradient methods, such as the one described in the previous section, improve $\eta(\pi)$ by taking a step on a local approximation at π_0:

$$L(\pi) \triangleq \eta(\pi_0) + \mathbb{A}_{\pi_0}(\pi) \quad (12)$$

where $\mathbb{A}_{\pi_0}(\pi) = \mathbb{E}_{s\sim\rho_{\pi_0}}\mathbb{E}_{a\sim\pi(\cdot|s)}[A_{\pi_0}(s,a)]$. If the policies are parameterized by θ (that is $\pi_0 = \pi_{\theta_0}$ and $\pi = \pi_\theta$), then L matches η to first order at θ_0, and therefore taking a small gradient step on L guarantees improvement of η. However, there is little guidance on how large this step can be, and in cases when the gradient can only be estimated, the required step size might be extremely small to compensate for noise. Schulman et al. (2015) address this by showing that minimizing a certain surrogate loss function can guarantee policy improvement with a large step size. Define the following penalized variant of L:

$$M(\pi) \triangleq L(\pi) + \frac{2\epsilon\gamma}{(1-\gamma)^2}\max_s D_{\mathrm{KL}}(\pi_0(\cdot|s) \parallel \pi(\cdot|s)) \quad (13)$$

where $\epsilon = \max_{s,a} |A_{\pi_0}(s,a)|$. Schulman et al. prove that M upper bounds η:

$$\eta(\pi) \leq M(\pi) \tag{14}$$

Because KL divergence is zero when its arguments are equal, this inequality shows that M majorizes [2] η at π_0. Using M as a majorizer for η in a majorization-minimization algorithm leads to an algorithm guaranteeing monotonic policy improvement at each iteration.

Unfortunately, as M is currently defined, computing the maximum-KL divergence term over the whole state space is intractable. Schulman et al. propose to relax this to an average over state space, which can be approximated by samples:

$$\overline{D}_{\mathrm{KL}}(\pi_0 \parallel \pi) \triangleq \mathbb{E}_{s\sim\rho_{\pi_0}} \left[D_{\mathrm{KL}}(\pi_0(\cdot|s) \parallel \pi(\cdot|s))\right] \tag{15}$$

They find that this average-KL formulation works well empirically, and that algorithm's stability could be improved by further reformulating the cost as a trust region constraint. This leads to the TRPO step computation

$$\underset{\theta}{\text{minimize}}\ L(\pi_\theta) \quad \text{s.t.} \quad \overline{D}_{\mathrm{KL}}(\pi_0 \parallel \pi_\theta) \leq \Delta \tag{16}$$

where all constants in Equation (13) are folded into a predefined trust region size $\Delta > 0$. To solve this step computation problem, the objective L and the KL divergence constraint must be approximated using samples and then minimized with gradient-based constrained optimization. The sample approximation can be done using a similar strategy to the one described in Section 4.1. Further discussion on sampling methodologies and effective optimization algorithms for solving this constrained problem can be found in Schulman et al. (2015).

TRPO for apprenticeship learning Now, we describe how to adapt TRPO to apprenticeship learning (3). Reintroducing the c superscripts, we wish to compute an improvement step from θ_0 to θ for the optimization problem

$$\underset{\theta}{\text{minimize}}\ \sup_{c\in\mathcal{C}} \eta^c(\pi_\theta) - \eta^c(\pi_E) \tag{17}$$

We wish to derive a majorizer for this objective, analogous to the majorizer (13) for a fixed cost function. Observe that if $\{f_\alpha\}$ and $\{g_\alpha\}$ are families of functions such that g_α majorizes f_α at x_0 for all α, then $\sup_\alpha g_\alpha$ majorizes $\sup_\alpha f_\alpha$ at x_0. We can therefore derive a TRPO-style algorithm for apprenticeship learning as follows. First, to remove the dependence of ϵ in (13) on any particular cost function, we assume that all cost functions in $\mathcal{C}$ are bounded by $C_{\max}$,[3] and then we let $\epsilon' \triangleq \frac{2C_{\max}}{1-\gamma} \geq \sup_c \max_{s,a} |A^c_{\pi_0}(s,a)|$. Now, we can define $M^c(\pi)$ analogously to (13):

$$M^c(\pi) \triangleq L^c(\pi) + \frac{2\epsilon'\gamma}{(1-\gamma)^2} \max_s D_{\mathrm{KL}}(\pi_0(\cdot|s) \parallel \pi(\cdot|s))$$

By the definition of ϵ' and (14), we have that for all $c \in \mathcal{C}$,

$$\eta^c(\pi) - \eta^c(\pi_E) \leq M^c(\pi) - \eta^c(\pi_E), \tag{18}$$

and consequently, we obtain an upper bound for the apprenticeship objective:

$$\begin{aligned}\delta_{\mathcal{C}}(\pi, \pi_E) &= \sup_{c\in\mathcal{C}} \eta^c(\pi) - \eta^c(\pi_E) \\ &\leq \sup_{c\in\mathcal{C}} M^c(\pi) - \eta^c(\pi_E) \triangleq M^{\mathcal{C}}(\pi, \pi_E).\end{aligned} \tag{19}$$

Since the inequalities (18) become equalities at $\pi = \pi_0$, inequality (19) does too, and thus $M^{\mathcal{C}}(\pi, \pi_E)$ majorizes the apprenticeship learning objective $\delta_{\mathcal{C}}(\pi, \pi_E)$ at $\pi = \pi_0$. Importantly, the KL divergence cost in $M^{\mathcal{C}}(\pi, \pi_E)$ does not depend on c:

$$\begin{aligned}M^{\mathcal{C}}(\pi, \pi_E) = &\left(\sup_{c\in\mathcal{C}} L^c(\pi) - \eta^c(\pi_E)\right) \\ &+ \frac{2\epsilon'\gamma}{(1-\gamma)^2} \max_s D_{\mathrm{KL}}(\pi_0(\cdot|s) \parallel \pi(\cdot|s))\end{aligned} \tag{20}$$

Hence, we can apply the same empirically justified transformation that led to TRPO: replacing the maximum-KL cost by an average-KL constraint. We therefore propose to compute steps for our apprenticeship learning setting by solving the following trust region subproblem:

$$\begin{aligned}\underset{\theta}{\text{minimize}} \quad & \sup_{c\in\mathcal{C}} L^c(\pi_\theta) - \eta^c(\pi_E) \\ \text{subject to} \quad & \overline{D}_{\mathrm{KL}}(\pi_0 \parallel \pi_\theta) \leq \Delta\end{aligned} \tag{21}$$

where again, all constants are folded into Δ. To solve this trust region subproblem in the finite-sample regime, we approximate the KL constraint using samples from π_0, just as TRPO does for its trust region problem (16). The objective of (21), however, warrants more attention, because of the interplay between the maximization over c and minimization over θ. Let $f(\theta)$ be the objective of (21). We wish to derive a finite-sample approximation to f suitable as an objective for the subproblem, so for computational reasons, we would like to avoid trajectory sampling within optimization for this subproblem.

To do so, we introduce importance sampling with π_0 as the proposal distribution for the advantage term of f, thereby avoiding the need to sample from π_θ as θ varies:

$$\begin{aligned}f(\theta) &= \sup_{c\in\mathcal{C}} \eta^c(\pi_0) - \eta^c(\pi_E) + \mathbb{A}^c_{\pi_0}(\pi_\theta) \\ &= \sup_{c\in\mathcal{C}} \mathbb{E}_{\rho_{\pi_0}}[c(s,a)] - \mathbb{E}_{\rho_{\pi_E}}[c(s,a)] + \\ &\qquad \mathbb{E}_{\rho_{\pi_0}}\left[\frac{\pi_\theta(a|s)}{\pi_0(a|s)}(Q^c_{\pi_0}(s,a) - V^c_{\pi_0}(s))\right]\end{aligned} \tag{22}$$

[2] M is said to majorize η at x_0 if $M \geq \eta$ with equality at x_0.

[3] In practice, this is easy to satisfy for $\mathcal{C}_{\text{linear}}$ and $\mathcal{C}_{\text{convex}}$ by ensuring that the cost basis functions are bounded.

Algorithm 2 IM-TRPO

Input: Expert trajectories τ_E, initial policy params. θ_0, trust region size Δ
for $i = 0, 1, 2, \ldots$ **do**
  Roll out trajectories $\tau \sim \pi_{\theta_i}$
  Find $\pi_{\theta_{i+1}}$ minimizing Equation (23)
    subject to $\overline{D}_{\mathrm{KL}}(\pi_{\theta_i} \,\|\, \pi_{\theta_{i+1}}) \leq \Delta$,
    with expectations taken over τ and τ_E (15)
end for

At first glance, it seems that the last term of equation (22) requires multiple rollouts for the Q and V parts separately. However, this is not the case, because the identity

$$\mathbb{E}_{a\sim\pi_0(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_0(a|s)}V^c_{\pi_0}(s)\right] = V^c_{\pi_0}(s) = \mathbb{E}_{a\sim\pi_0(\cdot|s)}[Q^c_{\pi_0}(s,a)]$$

lets us write

$$f(\theta) = \sup_{c\in\mathcal{C}} \mathbb{E}_{\rho_{\pi_0}}[c(s,a)] - \mathbb{E}_{\rho_{\pi_E}}[c(s,a)] + \mathbb{E}_{\rho_{\pi_0}}\left[\left(\frac{\pi_\theta(a|s)}{\pi_0(a|s)} - 1\right) Q^c_{\pi_0}(s,a)\right] \tag{23}$$

Replacing expectations with empirical ones gives the final form of the objective that we use to define the finite-sample trust region subproblem. Solving these trust region subproblems yields our final algorithm, which we call IM-TRPO (Algorithm 2). The computational power needed to minimize this trust region cost is not much greater than that of the TRPO subproblem (16), assuming that the supremum over $\mathcal{C}$ is easily computable. We will show next in Section 4.3 that solving IM-TRPO subproblems indeed poses no significant difficulty over the computation (10) necessary for IM-REINFORCE.

4.3. Finding cost functions

Until now, we deferred discussion of finding cost functions that achieve the supremum in the apprenticeship learning objective (2). We address this issue here for the feature expectation matching setting as described in Section 3.1, with cost functions $\mathcal{C}_{\text{linear}}$ parameterized linearly by ℓ_2-bounded weight vectors (4). The case for $\mathcal{C}_{\text{convex}}$ can be derived similarly and is omitted for space reasons.

As mentioned in Section 4.1, the apprenticeship learning algorithm only has access to sample trajectories from π and π_E, so we will consider finding the cost that achieves the supremum of the empirical apprenticeship learning objective $\hat{\delta}_{\mathcal{C}}$ (10). Using $\hat{c}$ to denote the optimal cost for this empirical objective, we have

$$\hat{\delta}_{\mathcal{C}_{\text{linear}}}(\pi, \pi_E) = \sup_{\|w\|_2\leq 1} w \cdot (\hat{\phi}(\pi) - \hat{\phi}(\pi_E)), \tag{24}$$

where $\hat{\phi}$ is the empirical feature expectation vector. The supremum in this equation is attained by a vector with a closed-form expression: $\hat{w} \triangleq (\hat{\phi}(\pi) - \hat{\phi}(\pi_E))/\|\hat{\phi}(\pi) - \hat{\phi}(\pi_E)\|$, which can be inserted directly into (8) for IM-REINFORCE. However, this $\hat{w}$ does not suffice for IM-TRPO, which, for the objective of the trust region subproblem (23), requires a maximizer $\hat{w}$ that must be recomputed for every optimization step for the subproblem. This recomputation is not difficult or expensive, as we will now demonstrate. For linear costs, the empirical trust region subproblem objective is given by:

$$\hat{f}(\theta) = \sup_{\|w\|_2\leq 1} \hat{\mathbb{E}}_{\rho_{\pi_0}}[w\cdot\phi(s,a)] - \hat{\mathbb{E}}_{\rho_{\pi_E}}[w\cdot\phi(s,a)] + \hat{\mathbb{E}}_{\rho_{\pi_0}}\left[\left(\frac{\pi_\theta(a|s)}{\pi_0(a|s)} - 1\right) Q^w_{\pi_0}(s,a)\right] \tag{25}$$

Now let $\phi(\pi_0|s_0,a_0) \triangleq \mathbb{E}_{\pi_0}\left[\sum_{t=0}^{\infty}\gamma^t\phi(s_t,a_t)\,|\,s_0,a_0\right]$ and $\psi(\pi_\theta) \triangleq \mathbb{E}_{\rho_{\pi_0}}\left[\left(\frac{\pi_\theta(a|s)}{\pi_0(a|s)} - 1\right)\phi(\pi_0|s,a)\right]$, both of which are readily estimated from the very same rollout trajectories from π_0 used to estimate expected costs. With these, we get $Q^w_{\pi_0}(s,a) = w\cdot\phi(\pi_0|s,a)$, which lets us write (25) as:

$$\hat{f}(\theta) = \sup_{\|w\|_2\leq 1} w\cdot\left(\hat{\phi}(\pi_0) - \hat{\phi}(\pi_E) + \hat{\psi}(\pi_\theta)\right) \tag{26}$$

This reveals that the supremum is achieved by

$$\hat{w} \triangleq \frac{\hat{\phi}(\pi_0) - \hat{\phi}(\pi_E) + \hat{\psi}(\pi_\theta)}{\|\hat{\phi}(\pi_0) - \hat{\phi}(\pi_E) + \hat{\psi}(\pi_\theta)\|}, \tag{27}$$

which is straightforward to compute. Note that this vector depends on θ; that is, it changes as the trust region subproblem (21) is optimized, and must be recomputed with each step of the algorithm for solving the trust region subproblem. However, by construction, all empirical expectations are taken with respect to π_0, which does not change as θ changes, and hence no simulations in the environment are required for these recomputations.

5. Experiments

We evaluated our approach in a variety of scenarios: finite gridworlds of varying sizes, the continuous planar navigation task of Levine and Koltun (2012), a family of continuous environments of varying numbers of observation features (Karpathy, 2015), and a variation of Levine & Koltun's highway driving simulation, in which the agent receives high-dimensional egocentric observation features.

In all of the continuous environments, we used policies constructed according to Schulman et al. (2015): the policies have Gaussian action distributions, with mean given by a multi-layer perceptron taking observations as input, and standard deviations given by an extra set of parameters. Details on the environments and training methodology are in the supplement.

Comparing against globally optimal methods As mentioned in Section 3.1, LPAL (Syed et al., 2008) finds a global optimum for the apprenticeship problem (3) with $\mathcal{C}_{\text{convex}}$ in finite state and action spaces. In contrast, our approach scales to high-dimensional spaces but is only guaranteed to find a local optimum of (3), as described in Section 4. To evaluate the quality of our local optima, we tested IM-REINFORCE, using $\mathcal{C}_{\text{convex}}$ to learn tabular Boltzmann policies with value iteration for exact gradient evaluation, against LPAL and a behavioral cloning baseline.

We evaluated the learned policies for the three algorithms on 64×64 gridworlds on varying amounts of expert data. In each trial, we randomly generated costs in the world, and we generated expert data by sampling behavior from an optimal policy computed with value iteration. To evaluate an algorithm, we computed the ratio of learned policy performance to the expert's performance. We also ran a timing test, in which we evaluated the computation time for each algorithm on varying gridworld sizes, with fixed dataset sizes, for 10 trials each.

The results are displayed in Figure 1. We found that despite our local optimality guarantee, IM-REINFORCE learned policies achieving at least 98% the performance of policies learned by LPAL, with similar sample complexity. IM-REINFORCE's training times also scaled favorably compared to LPAL. For a large gridworld with 65536 states, LPAL took on average 10 minutes to train with large variance across instantiations of the expert, whereas our algorithm consistently took around 4 minutes.

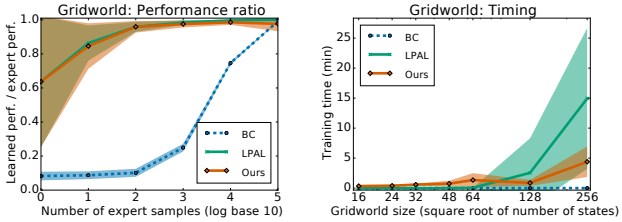

Figure 1. Left: Gridworld performance ratio across varying amounts of expert data. Right: Training time on increasing gridworld sizes. (*BC* stands for behavioral cloning.)

Comparing against continuous IRL Next, we evaluated our algorithms in a small, continuous environment: the objectworld environment of Levine and Koltun (2012), in which the agent moves in a plane to seek out Gaussian-shaped costs, given only expert data generated either by globally or locally optimal expert policies. We compared the trajectories produced by IM-TRPO with $\mathcal{C}_{\text{linear}}$ to those produced by trajectory optimization on a cost learned by Levine and Koltun's CIOC algorithm, a model-based IRL method designed for continuous settings with full knowledge of dynamics derivatives. The basis functions we used for $\mathcal{C}_{\text{linear}}$ were the same as those used by CIOC to define learned cost functions. The results are in Figure 2.

We found that even though our method is model-free and does not use dynamics derivatives, it consistently learned policies achieving zero excess cost (the difference in expected true cost compared to the expert, measured by averaging over 100 rollouts), matching the performance of optimal trajectories for cost functions learned by CIOC.

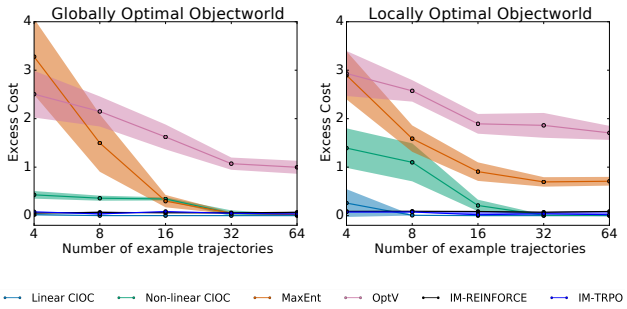

Figure 2. Excess cost for each algorithm for globally and locally optimal planar navigation examples, against variants of CIOC and other competing algorithms.

Varying dimension To evaluate our algorithms' performance with varying environment dimension, we used a family of environments inspired by Karpathy (2015). In these environments, the agent moves in a plane populated by colored moving targets to be either captured or avoided, depending on color. The action space is two-dimensional, allowing the agent to apply forces to move itself in any direction. The agent has a number of sensors N_{sensors} facing outward with uniform angular spacing. Each sensor detects the presence of a target with 5 continuous features indicating the nearest target's distance, color, and relative velocity to the agent. These sensor features, along with an indicators of whether the agent is currently capturing a target, lead to observation features of dimension $5 \cdot N_{\text{sensors}} + 2$, which are fed to the policy. Varying N_{sensors} yields a family of environments with differing observation dimension.

For N_{sensors} set to 5, 10, and 20 (yielding 27, 52, and 102 observation features, respectively), we first generated expert data by executing policies learned by reinforcement learning on a true cost, which was a linear combination of basis functions indicating control effort and intersection with targets. Then, we ran both IM-REINFORCE and IM-TRPO using $\mathcal{C}_{\text{linear}}$ on the same basis functions, and we measured the excess cost of each learned policy.

We found that IM-TRPO achieved nearly perfect imitation in this setting, and the performance was not significantly affected by the dimensionality of the space. IM-REINFORCE's learning also progressed, but was far outpaced by IM-TRPO (see Figure 3). We also verified that IM-TRPO's overhead of computing $\hat{c}$ (27) for each step of solving its trust region subproblem was negligible, verify-

ing our claim in Section 4.3. On our system, iterations for plain TRPO for reinforcement learning and IM-TRPO both took 8-9 seconds each for this environment, with no statistically significant difference.

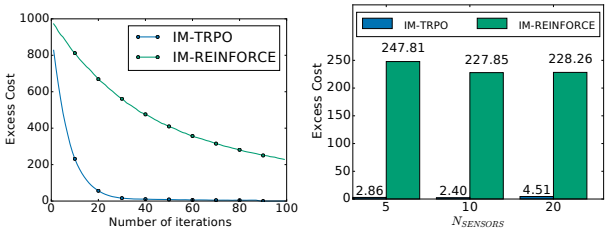

Figure 3. Left: Excess cost over time for one run of $N_{\text{sensors}} = 20$. Curves for other settings are similar. Right: Excess costs for learned policies on various sensor counts.

Highway driving Finally, we ran IM-TRPO on a variation of the highway driving task of Levine & Koltun (2012). In this task, the learner must imitate driving behaviors (aggressive, tailgating, and evasive) in a continuous driving simulation. The observations in the original driving task were the actual states of the whole environment, including positions and velocities of all cars at all points on the road. To introduce more realism, we modified the environment by providing policies only egocentric observation features: readings from 30 equally spaced rangefinders that detect the two road edges, readings from 60 equally spaced rangefinders that detect cars, and speed and angular velocity of the agent's car. These observations effectively form a depth image of nearby cars and lane markings within the agent's field of view; see Figure 4. We aggregated these readings over a window of 5 timesteps, yielding a 610-dimensional partial observations.

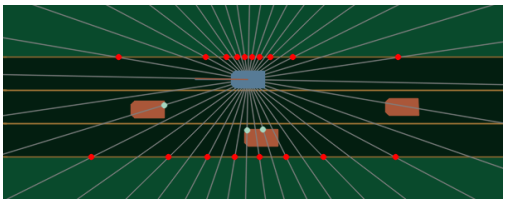

Figure 4. Car sensors for highway driving. The blue car is the agent, and emanating lines indicate sensor directions. Some sensors see lanes (red points), and others see neighboring cars (cyan points).

We ran IM-TRPO with $\mathcal{C}_{\text{linear}}$, using basis functions representing quadratic features derived from those of Levine & Koltun. Despite the fact that we provided only high-dimensional partial observations, our model-free approach learned policies achieving behavior comparable to the trajectories generated by CIOC, which was provided full state features and a full environment model. The policies learned by our algorithm generated behavior that both qualitatively and quantitatively resembled the demonstrated behavior, as shown in Table 1.

Table 1. Statistics for sample trajectories for IM-TRPO, compared to CIOC (Levine & Koltun, 2012) and human demonstrations. IM-TRPO and CIOC both generate human-like behavior.

STYLE	PATH	AVG. SPEED (KM/H)	TIME BEHIND (S)	TIME IN FRONT (S)
AGGRESSIVE	HUMAN	158.2	3.5	16.7
	CIOC	158.1	3.5	12.5
	IM-TRPO	147.8	4.2	9.2
EVASIVE	HUMAN	149.5	4.5	2.8
	CIOC	150.1	7.2	3.7
	IM-TRPO	110.4	4.6	3.9
TAILGATER	HUMAN	115.3	99.5	7.0
	CIOC	97.5	111.0	0.0
	IM-TRPO	97.6	71.4	12.3

6. Discussion and future work

We showed that carefully blending state-of-the-art policy gradient algorithms for reinforcement learning with local cost function fitting lets us successfully train neural network policies for imitation in high-dimensional, continuous environments. Our method is able to identify a locally optimal solution, even in settings where optimal planning is out of reach. This is a significant advantage over competing algorithms that require repeatedly solving planning problems in an inner loop. In fact, when the inner planning problem is only approximately solved, competing algorithms do not even provide local optimality guarantees (Ermon et al., 2015).

Our approach does not use expert interaction or reinforcement signal, fitting in a family of such approaches that includes apprenticeship learning and inverse reinforcement learning. When either of these additional resources is provided, alternative approaches (Kim et al., 2013; Daumé III et al., 2009; Ross & Bagnell, 2010; Ross et al., 2011) may be more sample efficient, and investigating ways to combine these resources with our framework is an interesting research direction.

We focused on the policy optimization component of apprenticeship learning, rather than the design of appropriate cost function classes. We believe this is an important area for future work. Nonlinear cost function classes have been successful in IRL (Ratliff et al., 2009; Levine et al., 2011) as well as in other machine learning problems reminiscent of ours, in particular that of training generative image models. In the language of generative adversarial networks (Goodfellow et al., 2014), the policy parameterizes a generative model of state-action pairs, and the cost function serves as an adversary. Apprenticeship learning with large cost function classes capable of distinguishing between arbitrary state-action visitation distributions would, enticingly, open up the possibility of exact imitation.

Acknowledgements

We thank John Schulman for valuable conversations about TRPO. This work was supported by a grant from the SAIL-Toyota Center for AI Research and by a National Science Foundation Graduate Research Fellowship (grant no. DGE-114747).

References

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

Bagnell, J Andrew. An invitation to imitation. Technical report, Carnegie Mellon University, 2015.

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

Ermon, Stefano, Xue, Yexiang, Toth, Russell, Dilkina, Bistra N, Bernstein, Richard, Damoulas, Theodoros, Clark, Patrick, DeGloria, Steve, Mude, Andrew, Barrett, Christopher, et al. Learning large-scale dynamic discrete choice models of spatio-temporal preferences with application to migratory pastoralism in East Africa. In *AAAI*, pp. 644–650, 2015.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Kakade, Sham and Langford, John. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, pp. 267–274, 2002.

Karpathy, Andrej. Reinforcejs: Waterworld demo, 2015. URL http://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html.

Kim, Beomjoon, Farahmand, Amir-massoud, Pineau, Joelle, and Precup, Doina. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pp. 2859–2867, 2013.

Levine, Sergey and Koltun, Vladlen. Continuous inverse optimal control with locally optimal examples. In *Proceedings of the 29th International Conference on Machine Learning*, pp. 41–48, 2012.

Levine, Sergey, Popovic, Zoran, and Koltun, Vladlen. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pp. 19–27, 2011.

Neu, Gergely and Szepesvári, Csaba. Training parsers by inverse reinforcement learning. *Mach. Learn.*, 77(2-3): 303–337, 11 April 2009.

Ng, Andrew Y and Russell, Stuart J. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 663–670, 2000.

Peters, Jan and Schaal, Stefan. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21 (4):682–697, 2008.

Pomerleau, Dean A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

Ratliff, Nathan D, Bagnell, J Andrew, and Zinkevich, Martin A. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 729–736, 2006.

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

Ross, Stéphane and Bagnell, Drew. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 661–668, 2010.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, Drew. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

Russell, Stuart. Learning agents for uncertain environments. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pp. 101–103, 1998.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael, and Moritz, Philipp. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1889–1897, 2015.

Sutton, Richard S, McAllester, David A, Singh, Satinder P, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063, 1999.

Syed, Umar and Schapire, Robert E. A game-theoretic approach to apprenticeship learning. In *Advances in Neural Information Processing Systems*, pp. 1449–1456, 2007.

Syed, Umar and Schapire, Robert E. A reduction from apprenticeship learning to classification. In *Advances in Neural Information Processing Systems*, pp. 2253–2261, 2010.

Syed, Umar, Bowling, Michael, and Schapire, Robert E. Apprenticeship learning using linear programming. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 1032–1039, 2008.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Ziebart, Brian D, Maas, Andrew L, Bagnell, J Andrew, and Dey, Anind K. Maximum entropy inverse reinforcement learning. In *AAAI*, pp. 1433–1438, 2008.

Model-Free Imitation Learning with Policy Optimization: Supplementary Material

Jonathan Ho HOJ@CS.STANFORD.EDU
Jayesh K. Gupta JKG@CS.STANFORD.EDU
Stefano Ermon ERMON@CS.STANFORD.EDU
Stanford University

Here, we give extra information regarding the environment and algorithm setups for our experiments.

Gridworld We used tabular policies for IM-REINFORCE with parameters θ_{sa}, with action probabilities $\pi(a|s) \propto \exp(\theta_{sa})$; we used value iteration to obtain Q values for the gradient formula (8). We solved the linear programs for LPAL with Gurobi 6.5.1, and we defined the policies learned by behavioral cloning as simple lookups into expert data (for states unseen in the expert data, a random action is chosen). All timing tests were performed on an 4-core 3.6GHz Intel i7-4790 CPU.
The gridworlds we used resembled those of Abbeel and Ng (2004). Each was a square grid of states, with five actions (an action to move in each compass direction, and one for staying in place) that fail with 30% probability and result in a random move. Each test consisted of 40 trials. Costs were generated in 8×8 non-overlapping regions in the gridworld, giving one basis function for $\mathcal{C}_{\text{convex}}$ per region.

Waterworld We first ran TRPO for various iteration counts to obtain expert policies achieving various expected costs according to the true cost function, which penalized application of control, and assigned differing cost values to the targets of different colors. Then, we executed each expert policy to yield 25 trajectory samples, and then we ran IM-REINFORCE and IM-TRPO both for 100 iterations to imitate each expert policy. The trajectories were 500 timesteps long, and the discount factor was 0.99. We gave both algorithms 50 rollouts per iteration. Excess costs were computed by averaging over 100 rollouts.

Highway For each driving style, we ran IM-TRPO for 500 iterations, each collecting 20000 state-action pairs with simulation. The datasets and dynamics model were identical to the ones used by Levine & Koltun (2012). We evaluated our policies with the same measurements used by Levine & Koltun, averaged over 50 rollouts.