Behavioral Cloning from Observation

Faraz Torabi[1], **Garrett Warnell**[2], **Peter Stone**[1]
[1] The University of Texas at Austin
[2] U.S. Army Research Laboratory
{faraztrb,pstone}@cs.utexas.edu, garrett.a.warnell.civ@mail.mil

Abstract

Humans often learn how to perform tasks via imitation: they observe others perform a task, and then very quickly infer the appropriate actions to take based on their observations. While extending this paradigm to autonomous agents is a well-studied problem in general, there are two particular aspects that have largely been overlooked: (1) that the learning is done from observation *only* (i.e., without explicit action information), and (2) that the learning is typically done very quickly. In this work, we propose a two-phase, autonomous imitation learning technique called *behavioral cloning from observation (BCO)*, that aims to provide improved performance with respect to both of these aspects. First, we allow the agent to acquire experience in a self-supervised fashion. This experience is used to develop a model which is then utilized to learn a particular task by observing an expert perform that task without the knowledge of the specific actions taken. We experimentally compare BCO to imitation learning methods, including the state-of-the-art, generative adversarial imitation learning (GAIL) technique, and we show comparable task performance in several different simulation domains while exhibiting increased learning speed after expert trajectories become available.

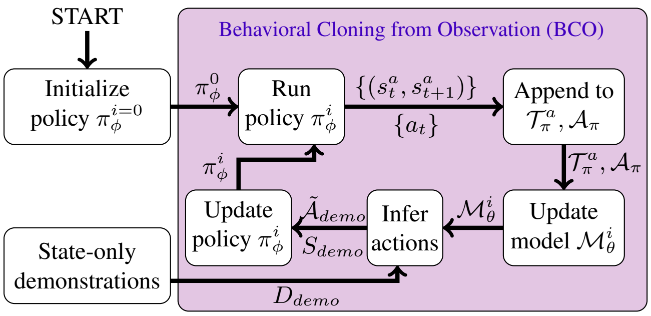

START
Behavioral Cloning from Observation (BCO)
Initialize policy $\pi_\phi^{i=0}$
π_ϕ^0
Run policy π_ϕ^i
$\{(s_t^a, s_{t+1}^a)\}$
$\{a_t\}$
Append to $\mathcal{T}_\pi^a, \mathcal{A}_\pi$
$\mathcal{T}_\pi^a, \mathcal{A}_\pi$
π_ϕ^i
Update policy π_ϕ^i
$\tilde{A}_{demo}$
S_{demo}
Infer actions
$\mathcal{M}_\theta^i$
Update model $\mathcal{M}_\theta^i$
State-only demonstrations
D_{demo}

Figure 1: Behavioral Cloning from Observation (BCO(α)) framework proposed in this paper. The agent is initialized with a (random) policy which interacts with the environment and collects data to to learn its own agent-specific inverse dynamics model. Then, given state-only demonstration information, the agent uses this learned model to infer the expert's missing action information. Once these actions have been inferred, the agent performs imitation learning. The updated policy is then used to collect data and this process repeats.

1 Introduction

The ability to learn through experience is a hallmark of intelligence. Humans most certainly learn this way, and, using *reinforcement learning (RL)* [Sutton and Barto, 1998], autonomous agents may do so as well. However, learning to perform a task based solely on one's own experience can be very difficult and slow. Humans are often able to learn much faster by observing and imitating the behavior of others. Enabling this same ability in autonomous agents, referred to as *learning from demonstration (LfD)*, has been given a great deal of attention in the research community [Schaal, 1997; Argall *et al.*, 2009].

While much of LfD research is motivated by the way humans learn from observing others, it has largely overlooked the integration of two important aspects of that paradigm. First, unlike the classical LfD setting, humans do not typically have knowledge of the precise *actions* executed by demonstrators, i.e., the internal control signals demonstrators use to guide their behavior Second, humans are able to perform imitation without needing to spend a lot of time interacting with their environment after the demonstration has been provided.

Most LfD work is unlike human imitation in its assumption that imitators know the actions executed by demonstrators. Human imitators typically do not have access to this information, and requiring it immediately precludes using a large amount of demonstration data where action sequences are not given. For example, there is a great number of tutorial videos on YouTube that only provide the observer knowledge of the demonstrator's state trajectory. It would be immensely beneficial if we could devise LfD algorithms to make use of such information.

Another challenge faced by LfD is the necessity of environment interaction, which can be expensive in several regards. One is the amount of time it requires: executing actions, either in the real world or in simulation, takes time. If a learning algorithm requires that a large number of actions must be executed in order to find a good imitation policy after a demonstration is presented, then there will be an undesirable amount of delay before the imitating agent will be successful. Furthermore, algorithms that require post-demonstration interaction typically require it again and again for each newly-demonstrated task, which could result in even more delay. Beyond delay, environment interaction can also be risky. For example, when training autonomous vehicles, operating on city streets while learning might endanger lives or lead to costly damage due to crashes. Therefore, we desire an algorithm for which environment interactions can be performed as a pre-processing step - perhaps in a safer environment - and where the information learned from those interactions can be re-used for a variety of demonstrated tasks.

In this paper, we propose a new imitation learning algorithm called *behavioral cloning from observation (BCO)*. BCO simultaneously addresses both of the issues discussed above, i.e., it provides reasonable imitation policies almost immediately upon observing state-trajectory-only demonstrations. First, it calls for the agent to learn a task-independent, inverse dynamics model in a pre-demonstration, exploratory phase. Then, upon observation of a demonstration without action information, BCO uses the learned model to infer the missing actions. Finally, BCO uses the demonstration and the inferred actions to find a policy via behavioral cloning. If post-demonstration environment interaction is allowed, BCO additionally specifies an iterative scheme where the agent uses the extra interaction time in order to learn a better model and improve its imitation policy. This iterative scheme therefore provides a tradeoff between imitation performance and post-demonstration environment interaction.

2 Related Work

BCO is related to both imitation learning and model-based learning. We begin with a review of imitation learning approaches, which typically fall under one of two broad categories: *behavioral cloning (BC)* and *inverse reinforcement learning (IRL)*.

Behavioral cloning [Bain and Sommut, 1999; Ross *et al.*, 2011; Daftry *et al.*, 2016] is one of the main methods to approach an imitation learning problem. The agent receives as training data both the encountered states and actions of the demonstrator, and then uses a classifier or regressor to replicate the expert's policy [Ross and Bagnell, 2010]. This method is powerful in the sense that it is capable of imitating the demonstrator immediately without having to interact with the environment. Accordingly, BC has been used in a variety of applications. For instance, it has been used to train a quadrotor to fly down a forest trail [Giusti *et al.*, 2016]. There, the training data is the pictures of the forest trail labeled with the actions that the demonstrating quadrotor used, and the policy is modeled as a convolutional neural network classifier. In the end, the quadrotor manages to fly down the trail successfully. BC has also been used in autonomous driving [Bojarski *et al.*, 2016]. The training data is acquired from a human demonstrator, and a convolutional neural network is trained to map raw pixels from a single front-facing camera directly to steering commands. After training, the vehicle is capable of driving in traffic on local roads. BC has also been successfully used to teach manipulator robots complex, multi-step, real-world tasks using kinesthetic demonstration [Niekum *et al.*, 2015]. While behavioral cloning is powerful, it is also only applicable in scenarios where the demonstrator's action sequences are available. However, when humans want to imitate each other's actions, they do not have access to the internal control signals the demonstrator used. Instead, they only see the effects of those actions. In our setting, we wish to perform imitation in these scenarios, and so we cannot apply BC techniques as they are typically formulated.

Inverse reinforcement learning is a second category of imitation learning. IRL techniques seek to learn a cost function that has the minimum value for the demonstrated actions. The learned cost function is then used in combination with RL methods to find an imitation policy. Like BC techniques, IRL methods usually assume that state-action pairs are available [Finn *et al.*, 2016; Ho and Ermon, 2016; Ho *et al.*, 2016], and also that the reward is a function of both states and actions. An exception is the work of Liu *et al.* [2017]. In this work, it is assumed that both demonstrator and imitator are capable of following a trajectory at the exact same pace to perform a task, and the IRL method defines the reward signal to be the proximity of the imitator and demonstrator's encoded state features at each time step. As a result, the reward signal can only be generated after the demonstration is made available, after which reinforcement learning and environment interaction must be completed in order to find a good policy. In our work, we wish to minimize the amount of environment interaction necessary after the demonstration is provided, and so we seek an alternative to IRL.

BCO is also related to the model-based learning literature in that it makes use of learned models of the environment. In general, model-based methods have major advantages over those that are model-free. First, they are more sample-efficient [Chebotar *et al.*, 2017], i.e., they do not require as many environment interactions as model-free methods. Second, the learned models can be transferred across tasks [Taylor *et al.*, 2008]. Typical model-learning techniques focus on obtaining an estimate of the transition dynamics model, i.e., a mapping from current state and action to the next state. In our work, on the other hand, we want the agent to learn a model of the environment that will help us infer missing actions, and therefore BCO learns a slightly-different inverse dynamics model, i.e., a mapping from state transitions to the actions [Hanna and Stone, 2017].

There has also been recent work done where inverse models have been used to perform imitation learning in the absence of action information. Niekum et al [2015a] present such a technique for situations in which the kinematic equations are known. Nair et al [2017] propose a technique that first learns an inverse dynamics model and then use that model to estimate the missing action information at each time step from a single demonstration. The method we develop here, on the other hand, both does not assume prior knowledge of the inverse model and is capable of generalizing in cases when mulitiple demonstrations are available.

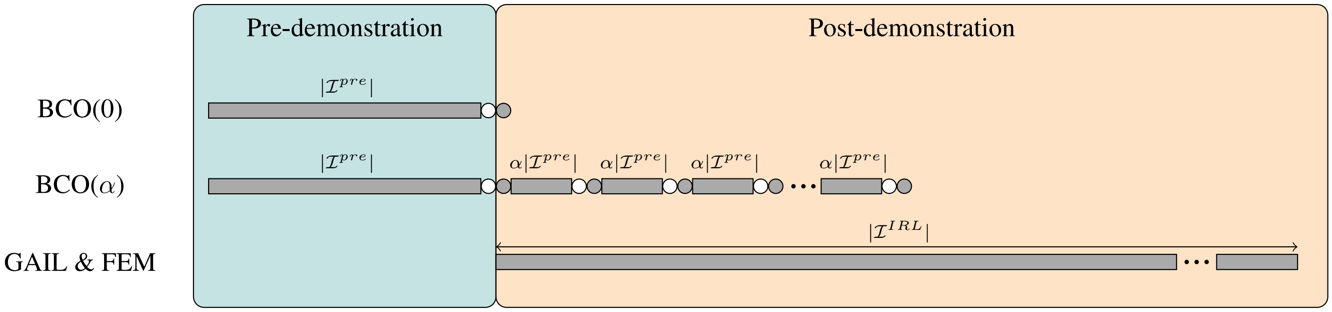


Figure 2: Learning timelines for BCO(0), BCO(α), and the IRL methods we compare against in this paper. The horizontal axis represents time, gray rectangles mark when each technique requires environment interactions. For BCO, the white and gray circles denote the inverse model and policy learning steps, respectively. For BCO, $\alpha|\mathcal{I}^{pre}|$ is the number of post-demonstration environment interactions performed before each model- and policy-improvement step and for IRL methods, $|\mathcal{I}^{IRL}|$ represents the total number of interactions.

3 Problem Formulation

We consider agents acting within the broad framework of Markov decision processes (MDPs). We denote a MDP using the 5-tuple $M = \{S, A, T, r, \gamma\}$, where S is the agent's state space, A is its action space, $T^{s_i a}_{s_{i+1}} = P(s_{i+1}|s_i, a)$ is a function denoting the probability of the agent transitioning from state s_i to s_{i+1} after taking action a, $r : S \times A \rightarrow \mathbb{R}$ is a function specifying the immediate reward that the agent receives for taking a specific action in a given state, and γ is a discount factor. In this framework, agent behavior can be specified by a policy, $\pi : S \rightarrow A$, which specifies the action (or distribution over actions) that the agent should use when in a particular state. We denote the set of state transitions experienced by an agent during a particular execution of a policy π by $\mathcal{T}_\pi = \{(s_i, s_{i+1})\}$.

In the context of these transitions, we will be interested in the *inverse dynamics model*, $\mathcal{M}^{s_i s_{i+1}}_a = P(a|s_i, s_{i+1})$, which is the probability of having taken action a given that the agent transitioned from state s_i to s_{i+1}. Moreover, we specifically seek task-independent models. We assume that some of the state features are specifically related to the task and others specifically to the agent, i.e., a given state s can be partitioned into an *agent-specific state*, s^a, and a *task-specific state*, s^t, which are members of sets S^a and S^t, respectively (i.e., $S = S^a \times S^t$) [Konidaris, 2006; Gupta *et al.*, 2017]. Using this partitioning, we define the *agent-specific inverse dynamics model* to be a function $\mathcal{M}_\theta : S^a \times S^a \rightarrow p(A)$ that maps a pair of agent-specific state transitions, $(s^a_i, s^a_{i+1}) \in \mathcal{T}^a_\pi$, to a distribution of agent actions that is likely to have given rise to that transition.

Imitation learning is typically defined in the context of a MDP *without* an explicitly-defined reward function, i.e., $M \setminus r$. The learning problem is for an agent to determine an *imitation policy*, $\pi : S \rightarrow A$ that the agent may use in order to behave like the expert, using a provided set of expert demonstrations $\{\xi_1, \xi_2, ...\}$ in which each ξ is a demonstrated state-action trajectory $\{(s_0, a_0), (s_1, a_1), ..., (s_N, a_N)\}$. Therefore, in this setting, the agent must have access to the demonstrator's actions. If the imitator is unable to observe the action sequences used by the demonstrator, the resulting imitation learning problem has recently been referred to as *imitation from observation* [Liu *et al.*, 2017]. In this setting, one seeks to find an imitation policy from a set of state-only demonstrations $D = \{\zeta_1, \zeta_2, ...\}$ in which each ζ is a state-only trajectory $\{s_0, s_1, ..., s_N\}$.

The specific problem that we are interested in is imitation from observation under a constrained number of environment interactions. By *environment interactions* we mean time steps for which we require our agent to gather new data by executing an action in its environment and observing the state outcome. We are concerned here in particular with the cost of the learning process, in terms of the number of environment interactions, both before and after the expert demonstrations are provided. Pre- and post-demonstration environment interactions are represented by $\mathcal{I}^{pre}$ and $\mathcal{I}^{post}$, respectively, to denote sets of interactions (s_i, a_i, s_{i+1}) that must be executed by a learner before and after a demonstration becomes available. In this context, we are concerned here with the following specific goal: *given a set of state-only demonstration trajectories, D, find a good imitation policy using a minimal number of post-demonstration environment interactions, i.e.,* $|\mathcal{I}^{post}|$.

In pursuit of this goal, we propose a new algorithm for imitation learning that can operate both in the absence of demonstrator action information and while requiring no or very few post-demonstration environment interactions. Our framework consists of two components, each of which considers a separate part of this problem. The first of these components considers the problem of learning an agent-specific inverse dynamics model, and the second one considers the problem of learning an imitation policy from a set of demonstration trajectories.

4 Behavioral Cloning from Observation

We now describe our imitation learning algorithm, BCO, which combines inverse dynamics model learning with learning an imitation policy. We are motivated by the fact that humans have access to a large amount of prior experience about themselves, and so we aim to also provide an autonomous agent with this same prior knowledge. To do so, before any demonstration information is observed, we allow the agent to learn its own agent-specific inverse dynamics model. Then, given state-only demonstration information, we use this learned model to infer the expert's missing action information. Once these actions have been inferred, the agent performs imitation learning via a modified version of behavioral cloning (Figure 1). The pseudo-code of the algorithm is given in Algorithm 1.

4.1 Inverse Dynamics Model Learning

In order to infer missing action information, we first allow the agent to acquire prior experience in the form of an agent-specific inverse dynamics model. In order to do so, we let the agent perform an exploration policy, π. In this work, we let π be a random policy (Algorithm 1, Line 2). While executing this policy, the agent performs some number of interactions with the environment, i.e., $\mathcal{I}^{pre}$. Because we seek an agent-specific inverse dynamics model as described in Section 3, we extract the agent-specific part of the states in $\mathcal{I}^{pre}$ and store them as $\mathcal{T}^a_{\pi_e} = \{(s^a_i, s^a_{i+1})\}$, and their associated actions, $\mathcal{A}_{\pi_e} = \{a_i\}$ (Algorithm 1, Lines 5-8). Given this information, the problem of learning an agent-specific inverse dynamics model is that of finding the parameter θ for which $\mathcal{M}_\theta$ best describes the observed transitions. We formulate this problem as one of maximum-likelihood estimation, i.e., we seek θ^* as

$$\theta^* = \arg\max_\theta \Pi_{i=0}^{|\mathcal{I}^{pre}|} p_\theta(a_i \,|s^a_i, s^a_{i+1}), \tag{1}$$

where p_θ is the conditional distribution over actions induced by $\mathcal{M}_\theta$ given a specific state transition. Any number of supervised learning techniques, denoted as "modelLearning" in Algorithm 1, may be used to solve (1).

Some details regarding particular choices made in this paper: For domains with a continuous action space, we assume a Gaussian distribution over each action dimension and our model estimates the individual means and standard deviations. We use a neural network for $\mathcal{M}_\theta$, where the network receives a state transition as input and outputs the mean for each action dimension. The standard deviation is also learned for each dimension, but it is computed independently of the state transitions. In order to train this network (i.e., to find θ^* in (1)), we use the Adam variant [Kingma and Ba, 2014] of stochastic gradient decent. Intuitively, the gradient for each sample is computed by finding a change in θ that would increase the probability of a_i with respect to the distribution specified by $\mathcal{M}_\theta(s_i, s_{i+1})$. When the action space is discrete, we again use a neural network for $\mathcal{M}_\theta$, where the network computes the probability of taking each action via a softmax function.

4.2 Behavioral Cloning

Our overarching problem is that of finding a good imitation policy from a set of state-only demonstration trajectories, $D_{demo} = \{\zeta_1, \zeta_2, \ldots\}$ where each ζ is a trajectory $\{s_0, s_1, \ldots, s_N\}$. Note that, although the inverse dynamics model is learned using a set of agent-generated data in Section 4.1, the data used there is not utilized in this step. In order to use the learned agent-specific inverse dynamics model, we first extract the agent-specific part of the demonstrated state sequences and then form the set of demonstrated agent-specific state transitions $\mathcal{T}^a_{demo}$ (Algorithm 1, Line 10). Next, for each transition $(s^a_i, s^a_{i+1}) \in \mathcal{T}^a_{demo}$, the algorithm computes the model-predicted distribution over demonstrator actions, $\mathcal{M}_{\theta^*}(s^a_i, s^a_{i+1})$ and uses the maximum-likelihood action as the inferred action, $\tilde{a}_i$, which is placed in a set $\tilde{\mathcal{A}}_{demo}$ (Algorithm 1, Line 11). Using these inferred actions, we then build the set of complete state-action pairs $\{(s_i, \tilde{a}_i)\}$.

Algorithm 1 BCO(α)

```
 1: Initialize the model $\mathcal{M}_\theta$ as random approximator
 2: Set $\pi_\phi$ to be a random policy
 3: Set $I = |\mathcal{I}^{pre}|$
 4: while policy improvement do
 5:     for time-step t=1 to $I$ do
 6:         Generate samples $(s^a_t, s^a_{t+1})$ and $a_t$ using $\pi_\phi$
 7:         Append samples $\mathcal{T}^a_{\pi_\phi} \leftarrow (s^a_t, s^a_{t+1})$, $\mathcal{A}_{\pi_\phi} \leftarrow a_t$
 8:     end for
 9:     Improve $\mathcal{M}_\theta$ by modelLearning($\mathcal{T}^a_{\pi_\phi}$, $\mathcal{A}_{\pi_\phi}$)
10:     Generate set of agent-specific state transitions $\mathcal{T}^a_{demo}$
        from the demonstrated state trajectories $D_{demo}$
11:     Use $\mathcal{M}_\theta$ with $\mathcal{T}^a_{demo}$ to approximate $\tilde{\mathcal{A}}_{demo}$
12:     Improve $\pi_\phi$ by behavioralCloning($S_{demo}$, $\tilde{\mathcal{A}}_{demo}$)
13:     Set $I = \alpha|\mathcal{I}^{pre}|$
14: end while
```

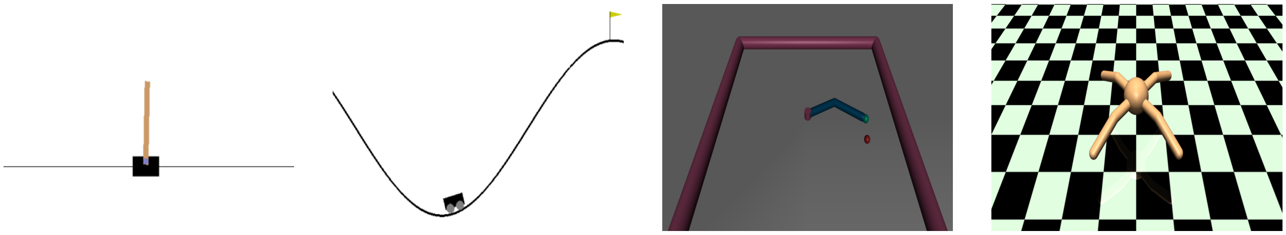
With this new set of state-action pairs, we may now seek the imitation policy π_ϕ. We cast this problem as one of behavioral cloning, i.e., given a set of state-action tuples $\{(s_i, \tilde{a}_i)\}$, the problem of learning an imitation policy becomes that of finding the parameter ϕ for which π_ϕ best matches this set of provided state-action pairs (Algorithm 1, Line 12). We find this parameter using maximum-likelihood estimation, i.e., we seek ϕ^* as

$$\phi^* = \arg\max_\phi \Pi_{i=0}^{N} \pi_\phi(\tilde{a}_i \mid s_i) \; . \tag{2}$$

Some details regarding particular choices made in this paper: For continuous action spaces, we assume our policy to be Gaussian over each action dimension, and, for discrete actions spaces we use a softmax function to represent the probability of selecting each value. We let π be a neural network that receives as input a state and outputs either the Gaussian distribution parameters or the action probabilities for continuous or discrete action spaces, respectively. We then solve for ϕ^* in (2) using Adam SGD, where the intuitive view of the gradient is that it seeks to find changes in ϕ that increase the probability of each inferred demonstrator action, $\tilde{a}_i$, in the imitation policy's distribution $\pi_\phi(\cdot \mid s_i)$.

4.3 Model Improvement

The techniques described above form the building blocks of BCO. If one is willing to tolerate post-demonstration environment interaction, a modified version of our algorithm can further improve both the learned model and the resulting imitation policy. This modified algorithm proceeds as follows. After the behavioral cloning step, the agent executes the imitation policy in the environment for a short period of time. Then, the newly-observed state-action sequences are used to update the model, and, accordingly, the imitation policy itself. The above procedure is repeated until there is



(a) CartPole (b) MountainCar (c) Reacher (d) Ant

Figure 3: Representative screenshots of the domains considered in this paper.

no more improvement in the imitation policy. We call this modified version of our algorithm BCO(α), where α is a user-specified parameter that is used to control the number of post-demonstration environment interactions at each iteration, M, according to $M = \alpha|\mathcal{I}^{pre}|$. The total number of post-demonstration interactions required by BCO(α) can be calculated as $|\mathcal{I}^{pre}| = TM = T\alpha|\mathcal{I}^{pre}|$, where T is the total number of model-improvement iterations required by BCO(α). Using a nonzero α, the model is able to leverage post-demonstration environment interaction in order to more accurately estimate the actions taken by the demonstrator, and therefore improve its learned imitation policy. If one has a fixed budget for post-demonstration interactions, one could consider terminating the model-improvement iterations early, i.e., specify both α and T.

5 Implementation and Experimental Results

We evaluated BCO(α) in several domains available in OpenAI Gym [Brockman *et al.*, 2016]. Continuous tasks are simulated by MuJoCo [Todorov *et al.*, 2012]. These domains have different levels of difficulty, as measured by the complexity of the dynamics and the size and continuity of the state and action spaces. Ordered from easy to hard, the domains we considered are: **CartPole**, **MountainCar**, **Reacher**, and **Ant-v1**. Each of these domains has predefined state features, actions and reward functions in the simulator. Any RL algorithm could be used with this reward function to generate an expert policy. Since trust region policy optimization (TRPO) [Schulman *et al.*, 2015] has shown promising performance in simulation [Liu *et al.*, 2017], we generated our demonstrations using agents trained with this method.

We evaluated our algorithm in two senses. First, with respect to the number of environment interactions required to attain a certain performance. In a real-world environment, interactions can be expensive which makes it a very important criterion. The second way in which we evaluate our algorithm is with respect to data efficiency, i.e., the imitator's task performance as a function of the amount of available demonstration data. In general, demonstration data is scarce, and so making the best use of it is very important.

We compared BCO(α) to the following methods:

1. **Behavioral Cloning (BC):** This method applies supervised learning over state-action pairs provided by the demonstrator.
2. **Feature Expectation Matching (FEM) [Ho *et al.*, 2016]:** A modified version of the approach presented by Abbeel and Ng [2004]. It uses trust region policy optimization with a linear cost function in order to train train neural network policies.
3. **General Adversarial Imitation Learning (GAIL) [Ho and Ermon, 2016]:** A state-of-the-art in IRL. It uses a specific class of cost functions which allows for the use of generative adversarial networks in order to do apprenticeship learning.

Note in particular that **our method is the only method that does not have access to the demonstrator's actions**. However, as our results will show, BCO(α) can still achieve comparable performance to these other techniques, and do so while requiring far fewer environment interactions.

5.1 Training Details and Results

Because both BC and BCO(α) rely on supervised learning methods, we use only 70% of the available data for training and use the rest for validation. We stop training when the error on the 30% validation data starts to increase. For the other methods, all available data was used in the training process. We will now discuss the architecture, details for each domain.

- **CartPole:** The goal is to keep the pole vertically upward as long as possible (Figure 3a). This domain has a discrete action space. In this domain, we considered linear models over the pre-defined state features for both the inverse dynamics model and the imitation policy and we only used $M = 1000$ interactions to learn the dynamics.
- **MountainCar:** The goal is to have the car reach the target point (Figure 3b). This domain has a discrete action space. In this domain, the data set for learning the inverse dynamics model is acquired by letting the agent to explore its action space for $M = 2000$ time steps. For both the imitation policy and inverse dynamics model, we used neural networks with two hidden layers, 8 nodes each, and leaky rectified linear activation functions (LReLU).
- **Reacher:** This domain has a continuous action space. The goal is to have the fingertip of the arm reach the target point whose position changes in every episode (Figure 3c). Therefore, in this domain, we can partition the state-space to agent-specific features (i.e., those only related to the arm) and task-specific features (i.e., those related to the position of the target). A neural network

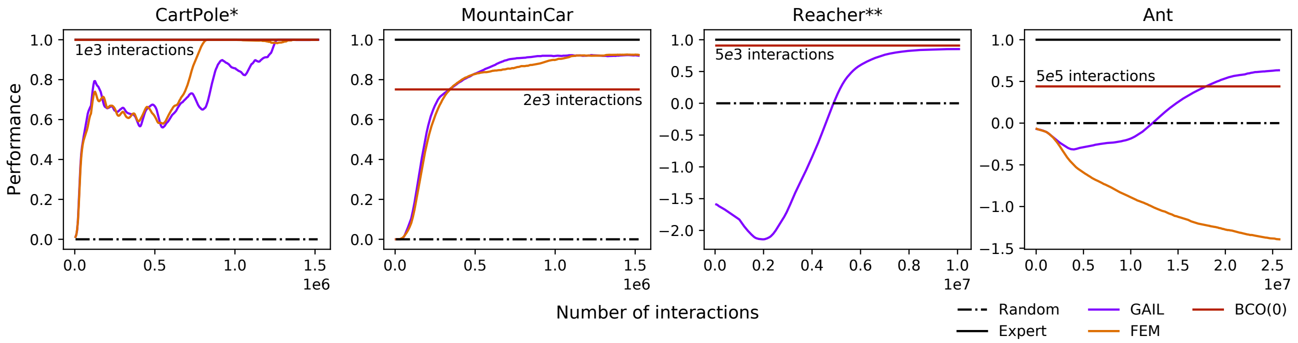


Figure 4: Performance of each technique with respect to the number of post-demonstration interactions. For each domain, ten demonstrated trajectories were considered. BCO(0) is depicted as a horizontal line since all environment interactions happen before the demonstration is provided. Performance values are scaled such that performance of a random policy is zero and the performance of the expert is one. Note that GAIL and FEM have access to demonstration action information whereas BCO does not. *The BCO line is not visible for the CartPole domain because BCO has the same performance as the expert. **FEM is not shown for the Reacher domain because its performance is much worse than the other techniques.

.

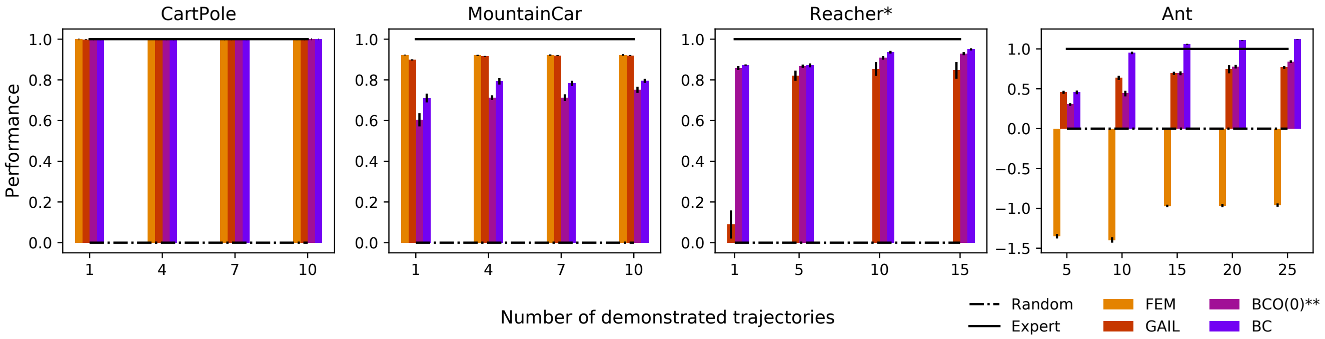


Figure 5: Performance of imitation agents with respect to the number of available demonstration trajectories. Rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 5000 trajectories. Returns have been scaled such that the performance of a random policy and the demonstrating agent are zero and one, respectively. *Note that FEM is not shown for the Reacher domain because its performance is much worse than the others. **Note that BC, GAIL, and FEM all have access to demonstration action information whereas *BCO(0) does not.*

architecture with two hidden layers of 100 LReLU nodes are used with $M = 5000$ agent-specific state transition-action pairs in order to learn the dynamics and then this model is used to learn a policy which also has two layers but with 32 LReLU nodes.

- **Ant:** The goal to have the ant to run as fast as possible(Figure 3d). This domain has a continuous action space. This is the most complex domain considered in this work. The state and action space are 111 and 8 dimensional, respectively. The number of interactions needed to learn the dynamics was $M = 5e5$ and the architectures for inverse dynamics learning and the policy are similar to those we used in **Reacher**.

5.2 Discussion

Each experiment was executed twenty times, and all results presented here are the average values over these twenty runs. We selected twenty trials because we empirically observed very small standard error bars in our results. This is likely a reflection of the relatively low variance in the expert demonstrations.

In our first experiment, we compare the number of environment interactions needed for BCO(0) with the number required by other methods (Figure 4.) We can clearly see how imitation performance improves as the agent is able to interact more with the environment. In the case of BCO(0), the interactions with the environment happen before the policy-learning process starts, and so we represent its performance with a horizontal line. The height of the line indicates the performance, and we display the number of pre-demonstration environment interactions it required next to it. The random and expert policies also do not benefit from post-demonstration environment interaction, and so they are also shown using horizontal lines. From these plots, we can see that it takes at least 40 times more interactions required by GAIL or FEM to gain the same performance as BCO(0).

Now, we aim to compare the performance of our algorithm BCO(α) with the other algorithms. To do so, we use

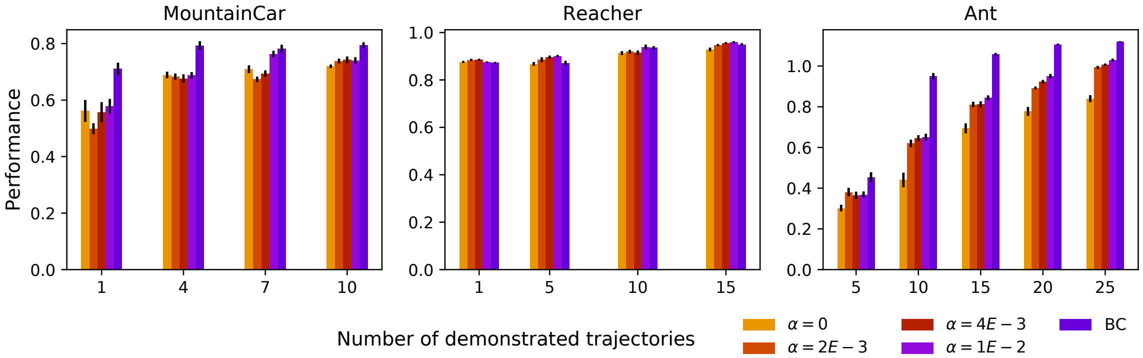


Figure 6: The performance of BC and several BCO(α) techniques (varying α) with respect to the number of demonstrated trajectories provided. Rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 5000 trajectories. By increasing α, more post-demonstration environment interactions are allowed to occur, which increases the performance of the imitation policy. Note that BC has access to demonstration action information whereas BCO does not. Also note that the number of trajectories required for learning a fairly good policy is very small. Each demonstrated trajectory has 5, 50, and 50 transitions for each domain from left to right, respectively. Note that we did not demonstrate the results for CartPole because the results were equally perfect regardless of the value of α.

MountainCar (pre-demo $= 2E3$)				**Reacher (pre-demo $= 5E3$)**				**Ant (pre-demo $= 5E5$)**			
d $\setminus \alpha$	**$2E-3$**	**$4E-3$**	**$1E-2$**	**d $\setminus \alpha$**	**$2E-3$**	**$4E-3$**	**$1E-2$**	**d $\setminus \alpha$**	**$2E-3$**	**$4E-3$**	**$1E-2$**
1	6825	23475	28950	**1**	210052	358736	912368	**5**	602500	1270000	3362500
4	8387	12000	31200	**5**	270500	486578	1837500	**10**	940000	2075000	5000000
7	6300	23100	122200	**10**	221421	569736	1055921	**15**	1387500	2855000	7325000
10	45462	61450	88600	**15**	509289	852210	1859210	**20**	1925000	4055000	9687500

Table 1: This is an extension to Figure 6 which provides the number of post-demonstration interactions for each case. First vertical column for each domain (d) shows the number of demonstrated trajectories. As an example, for the MountainCar domain with 1 demonstrated trajectory and $\alpha = 2E - 3$, the average number of post-demonstration interactions is 6825. We can also see at the top of the table that the number of pre-demonstration interactions is $2E3$ so the overall number of interactions would become 8825. It can be seen that almost always by increasing α or the number of demonstrated trajectories, the number of post-demonstration interactions increases. Also the overall number of interactions (combined pre- and post-demonstration interactions) in all the cases is far less than the overall number of interactions required by the other methods (FEM and GAIL).

each algorithm to train the agents, and then calculate the final performance by computing the average return over 5000 episodes. For comparison purposes, we scale these performance values such that the performance of the expert and a random policy are 1.0 and 0.0, respectively. This comparison is shown in Figures 5, and 6 where we have plotted the performance of each algorithm with respect to the number of available demonstrated trajectories. Figure 5 shows the comparison between the performance of BCO(0) with all the other methods, and Figure 6 compares the performance of BCO(α) across different values of α. In Figure 5, we can see that performance of our method is comparable with other methods even though our method is the only one without access to the actions. In the case of Reacher, the transferability of the learned inverse model is highlighted by the high performance of BCO. In the case of the Reacher and Ant domains, we can see that FEM performs poorly compared to others, perhaps because the rewards are not simple enough to be approximated by linear functions. In the CartPole domain, each of the methods performs as well as the expert and so all the lines are over each other. In the case of MountainCar, our performance is worse than other methods. Conversely, for Reacher, ours is more sample efficient than GAIL, i.e., with smaller number of demonstrations we get much better results. In the case of Ant, our method performs almost as good as GAIL. In Figure 6, we can see that BCO's performance improves with larger α since the extra environment interactions allow it to make better estimates of the demonstrator's actions.

6 Conclusions

In this paper, we have presented BCO, an algorithm for performing imitation learning that requires neither access to demonstrator actions nor post-demonstration environment interaction. Our experimental results show that the resulting imitation policies perform favorably compared to those generated by existing imitation learning approaches that *do* require access to demonstrator actions. Moreover, BCO requires fewer post-demonstration environment interactions than these other techniques, meaning that a reasonable imitation policy can be executed with less delay.

Acknowledgments

Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

[Argall *et al.*, 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[Bain and Sommut, 1999] Michael Bain and Claude Sommut. A framework for behavioural claning. *Machine Intelligence 15*, 15:103, 1999.

[Bojarski *et al.*, 2016] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Chebotar *et al.*, 2017] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. *arXiv preprint arXiv:1703.03078*, 2017.

[Daftry *et al.*, 2016] Shreyansh Daftry, J Andrew Bagnell, and Martial Hebert. Learning transferable policies for monocular reactive mav control. In *International Symposium on Experimental Robotics*, pages 3–11. Springer, 2016.

[Finn *et al.*, 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, 2016.

[Giusti *et al.*, 2016] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[Gupta *et al.*, 2017] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[Hanna and Stone, 2017] Josiah P Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *AAAI*, pages 3834–3840, 2017.

[Ho and Ermon, 2016] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[Ho *et al.*, 2016] Jonathan Ho, Jayesh Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pages 2760–2769, 2016.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Konidaris, 2006] George D Konidaris. A framework for transfer in reinforcement learning. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[Liu *et al.*, 2017] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *arXiv preprint arXiv:1707.03374*, 2017.

[Niekum *et al.*, 2015] Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

[Ross and Bagnell, 2010] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

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

[Schaal, 1997] Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[Taylor *et al.*, 2008] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.

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