Better-than-Demonstrator Imitation Learning via Automatically-Ranked Demonstrations

Daniel S. Brown, **Wonjoon Goo** and **Scott Niekum**
Department of Computer Science
University of Texas at Austin United States
{dsbrown, wonjoon, sniekum}@cs.utexas.edu

Abstract: The performance of imitation learning is typically upper-bounded by the performance of the demonstrator. While recent empirical results demonstrate that ranked demonstrations allow for better-than-demonstrator performance, preferences over demonstrations may be difficult to obtain, and little is known theoretically about when such methods can be expected to successfully extrapolate beyond the performance of the demonstrator. To address these issues, we first contribute a sufficient condition for better-than-demonstrator imitation learning and provide theoretical results showing why preferences over demonstrations can better reduce reward function ambiguity when performing inverse reinforcement learning. Building on this theory, we introduce Disturbance-based Reward Extrapolation (D-REX), a ranking-based imitation learning method that injects noise into a policy learned through behavioral cloning to automatically generate ranked demonstrations. These ranked demonstrations are used to efficiently learn a reward function that can then be optimized using reinforcement learning. We empirically validate our approach on simulated robot and Atari imitation learning benchmarks and show that D-REX outperforms standard imitation learning approaches and can significantly surpass the performance of the demonstrator. D-REX is the first imitation learning approach to achieve significant extrapolation beyond the demonstrator's performance without additional side-information or supervision, such as rewards or human preferences. By generating rankings automatically, we show that preference-based inverse reinforcement learning can be applied in traditional imitation learning settings where only unlabeled demonstrations are available.

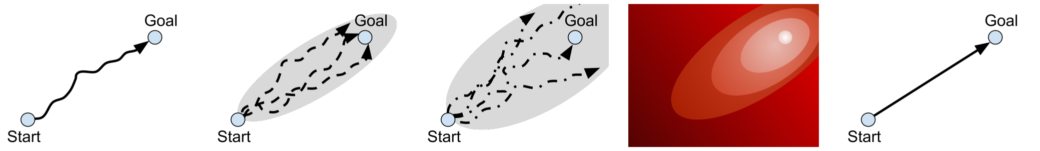
Keywords: imitation learning, reward learning, learning from preferences

1 Introduction

Imitation learning is a popular paradigm to teach robots and other autonomous agents to perform complex tasks simply by showing examples of how to perform the task. However, imitation learning methods typically find policies whose performance is upper-bounded by the performance of the demonstrator. While it is possible to learn policies that perform better than a demonstrator, existing methods either require access to a hand-crafted reward function [1] or a human supervisor who acts as a reward or value function during policy learning [2, 3]. Recent empirical results [3] give evidence that better-than-demonstrator performance can be achieved using ranked demonstrations; however, theoretical conditions for improvement over a demonstrator are lacking. This lack of theory makes it difficult to predict when current imitation learning approaches may exceed the performance of the demonstrator and precludes using theory to design better imitation learning algorithms.

In this paper, we first present theoretical results for when better-than-demonstrator performance is possible in an inverse reinforcement learning (IRL) setting [4], where the goal is to recover a reward function from demonstrations. We then present theoretical results demonstrating that rankings (or alternatively, pairwise preferences) over demonstrations can enable better-than-demonstrator performance by reducing error and ambiguity in the learned reward function. Next, we address the problem of leveraging the benefits of reward learning via ranked demonstrations in a way that does not require human rankings. Recently, Brown et al. [3] proposed Trajectory-ranked Reward

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.



(a) Demonstration (b) Small noise (c) Larger noise (d) Learned reward function from ranking: $(a) \succ (b) \succ (c)$ (e) Optimized policy

Figure 1: **D-REX high-level approach:** given a suboptimal demonstration (a), we run behavioral cloning to approximate the demonstrator's policy. By progressively adding more noise to this cloned policy ((b) and (c)), we are able to automatically synthesize a preference ranking: $(a) \succ (b) \succ (c)$. Using this ranking, we learn a reward function (d) which is then optimized using reinforcement learning to obtain a policy (e) that performs better than the demonstrator.

Extrapolation (T-REX), an imitation learning approach that uses a set of ranked demonstrations to learn a reward function that allows better-than-demonstrator performance without requiring human supervision during policy learning. However, requiring a demonstrator to rank demonstrations can be tedious and error prone, and precludes learning from prerecorded, unranked demonstrations, or learning from demonstrations of similar quality that are difficult to rank. Thus, we investigate whether it is possible to generate a set of ranked demonstrations, in order to surpass the performance of a demonstrator, without requiring supervised preference labels or reward information.

We propose Disturbance-based Reward Extrapolation (D-REX), a ranking-based reward learning algorithm that does not require ranked demonstrations. Our approach injects noise into a policy learned through behavioral cloning to automatically generate ranked policies of varying performance. D-REX makes the weak assumption that the demonstrations are better than a purely random policy, and that adding increasing levels of noise into a cloned policy will result in increasingly worse performance, converging to a random policy in the limit. Our approach is summarized in Figure 1. The intuition behind this approach is that generating ranked trajectories via noise injection reveals relative weightings between reward features: features that are more prevalent in noisier trajectories are likely inversely related to the reward, whereas features that are more common in noise-free trajectories are likely features which are positively correlated with the true reward. Furthermore, adding noise provides a form of feature selection since, if a feature is equally common across all levels of noise, then it likely has no impact on the true reward function and can be ignored.

By automatically generating rankings, preference-based imitation learning methods [5, 6, 7, 3] can be applied in standard imitation learning domains where rankings are unavailable. We demonstrate this by combining automatic rankings via noise-injections with a state-of-the-art imitation learning algorithm that uses ranked demonstrations [3]. We empirically validate our approach on simulated robotics and Atari benchmarks and find that D-REX results in policies that can both significantly outperform the demonstrator as well as significantly outperform standard imitation learning. To the best of our knowledge, D-REX is the first imitation learning approach to achieve significant performance improvements over the demonstrations without requiring extra supervision or additional side-information, such as ground-truth rewards or human preferences.

2 Related Work

Imitation learning has grown increasingly popular in recent years [8, 9], but little work has addressed the problem of achieving better-than-demonstrator performance. When ground-truth rewards are known, it is common to initialize a policy using demonstrations and then improve this policy using reinforcement learning [10, 1]. However, designing good reward functions for reinforcement learning can be difficult and can easily lead to unintended behaviors [11, 12].

Inverse reinforcement learning can be used to estimate a demonstrator's intent by inferring a reward function that explains the demonstrations. While there has been some work on learning from suboptimal demonstrations, most approaches either require suboptimal demonstrations to be manually clustered [13] or labeled [14]. Other methods are robust to unlabeled, suboptimal demonstrations, but require most demonstrations to come from an expert in order to correctly identify which demonstrations are anomalous [15, 16]. Syed and Schapire [17] proved that knowing which features contribute positively or negatively to the true reward allows an apprenticeship policy to outperform the demonstrator. However, their approach requires hand-crafted, linear features, knowledge of the true signs of the rewards features, and repeatedly solving a Markov decision process.

Preference learning [18] is another way to potentially learn better-than-demonstrator policies. Sadigh et al. [5] and Christiano et al. [2] propose reward learning approaches that use active learning to collect pairwise preferences labels. Ibarz et al. [6] and Palan et al. [7] combine demonstrations with active preference learning during policy optimization. Rather than collecting pairwise preferences via active queries, Brown et al. [3] propose Trajectory-ranked Reward Extrapolation (T-REX), an algorithm that uses a set of pre-ranked demonstrations to learn a reward function for policy optimization. Brown et al. evaluate T-REX on a variety of MuJoCo and Atari benchmarks and show that policies optimized via T-REX are able to consistently surpass the performance of a suboptimal demonstrator, but provide no theory to shed light on when extrapolation is possible. Our work provides a better theoretical understanding of when better-than-demonstrator performance is possible and why ranked demonstrations can help. Furthermore, our work demonstrates for the first time that ranking-based imitation learning approaches are applicable even in cases where human rankings are unavailable.

Prior work on imitation learning has investigated the use of random or noisy trajectories. Boularias et al. [19] and Kalakrishnan et al. [20] use uniformly random and locally perturbed trajectories, respectively, to estimate the partition function for Maximum Entropy inverse reinforcement learning [21]. Both methods seek a linear combination of predefined features such that the returns of the demonstrations are maximized with respect to the random trajectories. These methods can be seen as a special case of our proposed method, where only one level of noise is used and where the reward function is represented as a linear combination of known features. Disturbances for Augmenting Robot Trajectories (DART) [22] is a recently proposed behavioral cloning approach that adds noise during demonstrations to collect a richer set of state-action pairs for behavioral cloning. DART avoids the problem of compounding error that is common to most behavioral cloning approaches by repeatedly requesting and perturbing new demonstrations. Instead of repeatedly collecting perturbed trajectories from the demonstrator, we instead propose to collect a small number of initial demonstrations, run behavioral cloning once, and then inject varying amounts of noise into the cloned policy. This automatically creates a large set of ranked demonstrations for reward learning, without requiring a human to provide preference labels.

3 Problem Statement

Our goal is to achieve better-than-demonstrator performance via imitation learning. We model the environment as a Markov decision process (MDP) consisting of a set of states $\mathcal{S}$, actions $\mathcal{A}$, transition probabilities $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $R^* : \mathcal{S} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1)$. A policy π is a probability distribution over actions given state. Given a policy and an MDP, the expected discounted return of the policy is given by $J(\pi|R^*) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)]$. Similarly, the return of a trajectory consisting of states and actions, $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$, is given by $J(\tau|R^*) = \sum_{t=0}^{T} \gamma^t R^*(s_t)$.

We assume that we have no access to the true reward function of the MDP. Instead, we are given a set of m demonstrations $\mathcal{D} = \{\tau_1, \ldots \tau_m\}$, where each demonstrated trajectory is a sequence of states and actions, $\tau_i = (s_0, a_0, s_1, a_1, \ldots)$. We assume that the demonstrator is attempting (possibly unsuccessfully) to follow a policy that optimizes the true reward function R^*. Given the demonstrations $\mathcal{D}$, we wish to find a policy $\hat{\pi}$ that can extrapolate beyond the performance of the demonstrator. We say a policy $\hat{\pi}$ can extrapolate beyond of the performance of the demonstrator if it achieves a larger expected return than the demonstrations, when evaluated under the true reward function R^*, i.e., $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$, where $J(\mathcal{D}|R^*) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} J(\tau|R^*)$. Similarly, we say that a learned policy $\hat{\pi}$ extrapolates beyond the performance of the best demonstration if $J(\hat{\pi}|R^*) > \max_{\tau \in \mathcal{D}} J(\tau|R^*)$.

4 Extrapolating Beyond a Demonstrator: Theory

We first provide a sufficient condition under which it is possible to achieve better-than-demonstrator performance in an inverse reinforcement learning (IRL) setting, where the goal is to recover the demonstrator's reward function which is then used to optimize a policy [9]. We consider a learner that approximates the reward function of the demonstrator with a linear combination of features: $R(s) = w^T \phi(s)$.[1] These can be arbitrarily complex features, such as the activations of a deep neural network. The expected return of a policy when evaluated on $R(s)$ is given by

$$J(\pi|R) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right] = w^T \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \right] = w^T \Phi_\pi, \tag{1}$$

where Φ_π are the expected discounted feature counts that result from following the policy π.

Theorem 1. *If the estimated reward function is $\hat{R}(s) = w^T \phi(s)$, the true reward function is $R^*(s) = \hat{R}(s) + \epsilon(s)$ for some error function $\epsilon : \mathcal{S} \to \mathbb{R}$, and $\|w\|_1 \leq 1$, then extrapolation beyond the demonstrator, i.e., $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$, is guaranteed if :*

$$J(\pi^*_{R^*}|R^*) - J(\mathcal{D}|R^*) > \epsilon_\Phi + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{2}$$

where $\pi^*_{R^*}$ is the optimal policy under R^*, $\epsilon_\Phi = \|\Phi_{\pi^*_{R^*}} - \Phi_{\hat{\pi}}\|_\infty$ and $\|\epsilon\|_\infty = \sup\{\, |\epsilon(s)| : s \in \mathcal{S} \,\}$.

All proofs are given in the appendix. Intuitively, extrapolation depends on the demonstrator being sufficiently suboptimal, the error in the learned reward function being sufficiently small, and the state occupancy of the imitation policy, $\hat{\pi}$, being sufficiently close to $\pi^*_{R^*}$. If we can perfectly recover the reward function, then reinforcement learning can be used to ensure that ϵ_Φ is small. Thus, we focus on improving the accuracy of the learned reward function via automatically-ranked demonstrations. The learned reward function can then be optimized with any reinforcement learning algorithm [23].

4.1 Extrapolation via ranked demonstrations

The previous results demonstrate that in order to extrapolate beyond a suboptimal demonstrator, it is sufficient to have small reward approximation error and a good policy optimization algorithm. However, the following proposition, adapted from [24], shows that the reward function learned by standard IRL may be quite superficial and miss potentially important details, whereas enforcing a ranking over trajectories leads to a more accurate estimate of the true reward function.

Proposition 1. *There exist MDPs with true reward function R^*, expert policy π_E, approximate reward function $\hat{R}$, and non-expert policies π_1 and π_2, such that*

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|R^*) \;\; \textit{and} \;\; J(\pi_1|R^*) \ll J(\pi_2|R^*) \tag{3}$$

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \;\; \textit{and} \;\; J(\pi_1|\hat{R}) = J(\pi_2|\hat{R}). \tag{4}$$

However, enforcing a preference ranking over trajectories, $\tau^* \succ \tau_2 \succ \tau_1$, where $\tau^* \sim \pi^*$, $\tau_2 \sim \pi_2$, and $\tau_1 \sim \pi_1$, results in a learned reward function $\hat{R}$, such that

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \;\; \textit{and} \;\; J(\pi_1|\hat{R}) < J(\pi_2|\hat{R}). \tag{5}$$

Proposition 1 proves the existence of MDPs where an approximation of the true reward leads to an optimal policy, yet the learned reward reveals little about the underlying reward structure of the MDP. This is problematic for several reasons. The first problem is that if the learned reward function is drastically different than the true reward, this can lead to poor generalization. Another problem is that many learning from demonstration methods are motivated by providing *non-experts* the ability to program by example. Some non-experts will be good at personally performing a task, but may struggle when giving kinesthetic demonstrations [25] or teleoperating a robot [26, 27]. Other non-experts may not be able to personally perform a task at a high level of performance due to lack of precision or timing, or due to physical limitations or impairment. Thus, the standard IRL approach of finding a reward function that maximizes the likelihood of the demonstrations may lead to an incorrect, superficial reward function that overfits to suboptimal user behavior in the demonstrations.

Indeed, it has been proven that it is impossible to recover the correct reward function without additional information beyond observations, regardless of whether the policy is optimal [28] or

[1] Our results also hold for reward functions of the form $R(s, a) = w^T \phi(s, a)$.

suboptimal [29]. As demonstrated in Proposition 1, preference rankings can help to alleviate reward function ambiguity. If the true reward function is a linear combination of features, then the feasible region of all reward functions that make a policy optimal can be defined as an intersection of half-planes [30]: $H_\pi = \bigcap_{\pi' \in \Pi} w^T(\Phi_\pi - \Phi_{\pi'}) \geq 0$. We define the *reward ambiguity*, $G(H_\pi)$, as the volume of this intersection of half-planes: $G(H_\pi) = \text{Volume}(H_\pi)$, where we assume without loss of generality that $\|w\| \leq 1$, to ensure this volume is bounded. In Appendix A.2 we prove that a total ranking over policies results in less reward ambiguity than performing IRL on the optimal policy.

Proposition 2. *Given a policy class Π, an optimal policy $\pi^* \in \Pi$ and a total ranking over Π, and true reward function $R^*(s) = w^T\phi(s)$, the reward ambiguity resulting from π^* is greater than or equal to the reward ambiguity of using a total ranking, i.e., $G(H^*_\pi) \geq G(H_{\text{ranked}})$.*

Learning a reward function that respects a set of strictly ranked demonstrations avoids some of the ill-posedness of IRL [28] by eliminating a constant, or all-zero reward function. Furthermore, ranked demonstrations provide explicit information about both what to do as well as what *not* to do in an environment and each pairwise preference over trajectories gives a half-space constraint on feasible reward functions. In Appendix A.3 we prove that sampling random half-space constraints results in an exponential decrease in reward function ambiguity.

Corollary 1. *To reduce reward function ambiguity by x% it suffices to have $k = \log_2(1/(1-x/100))$ random half-space constraints over reward function weights.*

In practice, sampling random half-space constraints on the ground-truth reward function is infeasible. Instead, our proposed approach for better-than-demonstrator imitation learning uses noise injection to produce a wide variety of automatically-ranked of demonstrations in order to reduce the learner's reward function ambiguity. As we show in the next section, automatically-generating preferences over demonstrations also improves the efficiency of IRL by removing the need for an MDP solver in the inner-loop and turning IRL into a supervised learning problem [3].

5 Algorithm

We now describe our approach for achieving better-than-demonstrator imitation learning without requiring human-provided preference labels. We first briefly review a recent state-of-the-art IRL algorithm that learns from ranked demonstrations. We then describe our proposed approach to generate these rankings automatically via noise injection. Videos and code are available online.[2]

5.1 Trajectory-ranked Reward Extrapolation (T-REX)

Given a sequence of m demonstrations ranked from worst to best, $\tau_1, \ldots, \tau_m$, T-REX [3] first performs reward inference by approximating the reward at state s using a neural network, $\hat{R}_\theta(s)$, such that $\sum_{s\in\tau_i} \hat{R}_\theta(s) < \sum_{s\in\tau_j} \hat{R}_\theta(s)$ when $\tau_i \prec \tau_j$. The reward function $\hat{R}_\theta$ is learned via supervised learning, using a pairwise ranking loss [31] based on the Luce-Shephard choice rule [32]:

$$\mathcal{L}(\theta) \approx -\frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log \frac{\exp \sum_{s\in\tau_j} \hat{R}_\theta(s)}{\exp \sum_{s\in\tau_i} \hat{R}_\theta(s) + \exp \sum_{s\in\tau_j} \hat{R}_\theta(s)}, \tag{6}$$

where $\mathcal{P} = \{(i,j) : \tau_i \prec \tau_j\}$. After learning a reward function, T-REX can be combined with any RL algorithm to optimize a policy $\hat{\pi}$ with respect to $\hat{R}_\theta(s)$. Brown et al. [3] demonstrated that T-REX typically results in policies that perform significantly better than the best demonstration.

5.2 Disturbance-based Reward Extrapolation (D-REX)

We now describe Disturbance-based Reward Extrapolation (D-REX), our proposed approach for automatically generating ranked demonstrations. Our approach is summarized in Algorithm 1. We first take a set of unranked demonstrations and use behavioral cloning to learn a policy π_{BC}. Behavioral

[2] The project website and code can be found at https://dsbrown1331.github.io/CoRL2019-DREX/

Algorithm 1 D-REX: Disturbance-based Reward Extrapolation

```
Require: Demonstrations $\mathcal{D}$, noise schedule $\mathcal{E}$, number of rollouts $K$
Run behavioral cloning on demonstrations $\mathcal{D}$ to obtain policy $\pi_{\text{BC}}$
for $\epsilon_i \in \mathcal{E}$ do
    Generate a set of $K$ trajectories from a noise injected policy $\pi_{\text{BC}}(\cdot|\epsilon_i)$
end for
Generate automatic preference labels $\tau_i \prec \tau_j$ if $\tau_i \sim \pi_{\text{BC}}(\cdot|\epsilon_i)$, $\tau_j \sim \pi_{\text{BC}}(\cdot|\epsilon_j)$, and $\epsilon_i > \epsilon_j$
Run T-REX [3] on automatically ranked trajectories to obtain $\hat{R}$
Optimize policy $\hat{\pi}$ using reinforcement learning with reward function $\hat{R}$
return $\hat{\pi}$
```

cloning [33] treats each state action pair $(s, a) \in \mathcal{D}$ as a training example and seeks a policy π_{BC} that maps from states to actions. We model π_{BC} using a neural network with parameters θ_{BC} and find these parameters using maximum-likelihood estimation such that $\theta_{\text{BC}} = \arg\max_\theta \prod_{(s,a)\in\mathcal{D}} \pi_{\text{BC}}(a|s)$. By virtue of the optimization procedure, π_{BC} will usually only perform as well as the average performance of the demonstrator—at best it may perform slightly better than the demonstrator if the demonstrator makes mistakes approximately uniformly at random.

Our main insight is that if π_{BC} is significantly better than the performance of a completely random policy, then we can inject noise into π_{BC} and interpolate between the performance of π_{BC} and the performance of a uniformly random policy. In Appendix B.1, we prove that given a noise schedule $\mathcal{E} = (\epsilon_1, \epsilon_2, \ldots, \epsilon_d)$ consisting of a sequence of noise levels such that $\epsilon_1 > \epsilon_2 > \ldots > \epsilon_d$, then with high-probability, $J(\pi_{\text{BC}}(\cdot|\epsilon_1)) < J(\pi_{\text{BC}}(\cdot|\epsilon_2)) < \cdots < J(\pi_{\text{BC}}(\cdot|\epsilon_d))$. Given noise level $\epsilon \in \mathcal{E}$, we inject noise via an ϵ-greedy policy such that with probability 1-ϵ, the action is chosen according to π_{BC}, and with probability ϵ, the action is chosen uniformly at random within the action range.

For every ϵ, we generate K policy rollouts and thus obtain $K \times d$ ranked demonstrations, where each trajectory is ranked based on the noise level that generated it, with trajectories considered of equal preference if generated from the same noise level. Thus, by generating rollouts from $\pi_{\text{BC}}(\cdot|\epsilon)$ with varying levels of noise, we can obtain an arbitrarily large number of ranked demonstrations:

$$D_{\text{ranked}} = \{\tau_i \prec \tau_j : \tau_i \sim \pi_{\text{BC}}(\cdot|\epsilon_i), \tau_j \sim \pi_{\text{BC}}(\cdot|\epsilon_j), \epsilon_i > \epsilon_j\}. \tag{7}$$

Given these ranked demonstrations, we then use T-REX to learn a reward function $\hat{R}$ from which we can optimize a policy $\hat{\pi}$ using any reinforcement learning algorithm (see Appendix E for details).

6 Experimental Results

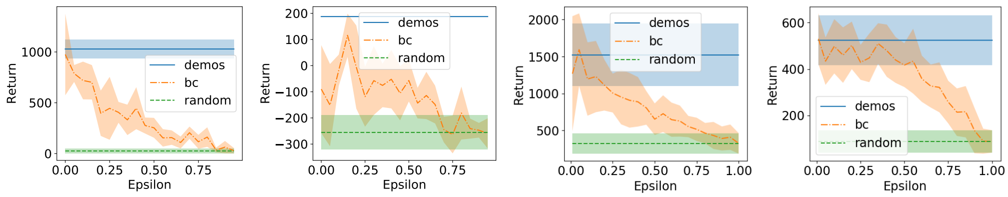
6.1 Automatically generating rankings via noise

To test whether injecting noise can create high-quality, automatic rankings, we used simulated suboptimal demonstrations from a partially trained reinforcement learning agent. To do so, we used the Proximal Policy Optimization (PPO) implementation from OpenAI Baselines to partially train a policy on the ground-truth reward function. We then ran behavioral cloning on these demonstrations and plotted the degradation in policy performance for increasing values of ϵ.

We evaluated noise degradation on the Hopper and Half-Cheetah domains in MuJoCo and on the seven Atari games listed in Table 1. To perform behavioral cloning, we used one suboptimal demonstration trajectory of length 1,000 for the MuJoCo tasks and 10 suboptimal demonstrations for the Atari games. We then varied ϵ and generated rollouts for different noise levels. We plotted the average return along with one standard deviation error bars in Figure 2 (see Appendix B for details). We found that behavioral cloning with small noise tends to have performance similar to that of the average performance of the demonstrator. As noise is added, the performance degrades until it reaches the level of a uniformly random policy ($\epsilon = 1$). These plots validate our assumption that, in expectation, adding increasing amounts of noise will cause near-monotonic performance degradation.

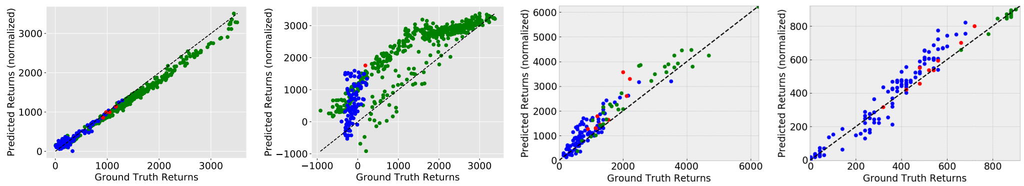
6.2 Reward extrapolation

We next tested whether D-REX allows for accurate reward extrapolation. We used noise injection, as described in the previous section, to generate 100 synthetically-ranked demonstrations. For



(a) Hopper (b) Half-Cheetah (c) Beam Rider (d) Seaquest

Figure 2: Examples of the degradation in performance of an imitation policy learned via behavioral cloning as more noise is injected into the policy. Behavioral cloning is done on a 1,000-length trajectory (MuJoCo tasks) or 10 demonstrations (Atari games). Plots show mean and standard deviations over 5 rollouts (MuJoCo tasks) or 20 rollouts (Atari games).



(a) Hopper (b) Half-Cheetah (c) Beam Rider (d) Seaquest

Figure 3: Extrapolation plots for a selection of MuJoCo and Atari tasks (see the appendix for more plots). Blue dots represent synthetic demonstrations generated via behavioral cloning with different amounts of noise injection. Red dots represent actual demonstrations, and green dots represent additional trajectories not seen during training. We compare ground truth returns over demonstrations to the predicted returns from D-REX (normalized to be in the same range as the ground truth returns).

MuJoCo, we used the noise schedule consisting of 20 different noise levels, evenly spaced over the interval $[0, 1)$ and generated $K = 5$ rollouts per noise level. For Atari, we used the noise schedule $\mathcal{E} = (1.0, 0.75, 0.5, 0.25, 0.02)$ with $K = 20$ rollouts per noise level. By automatically generating ranked demonstrations, D-REX is able to leverage a small number of unranked demonstrations to generate a large dataset of ranked demonstrations for reward function approximation. We used the T-REX algorithm [3] to learn a reward function from these synthetically ranked demonstrations.

To investigate how well D-REX learns the true reward function, we evaluated the learned reward function $\hat{R}_\theta$ on the original demonstrations and the synthetic demonstrations obtained via noise injection. We then compared the ground-truth returns with the predicted returns under $\hat{R}_\theta$. We also tested reward extrapolation on a held-out set of trajectories obtained from PPO policies that were trained longer on the ground-truth reward than the policy used to generate the demonstrations for D-REX. These additional trajectories allow us to measure how well the learned reward function can extrapolate beyond the performance of the original demonstrations. The results for four of the tasks are shown in Figure 3. The remaining plots are included in Appendix G. The plots show relatively strong correlation between ground truth returns and predicted returns across most tasks, despite having no a priori access to information about true returns, nor rankings. We also generated reward sensitivity heat maps [34] for the learned reward functions. These visualizations provide evidence that D-REX learns semantically meaningful features that are highly correlated with the ground truth reward. For example, on Seaquest, the reward function learns a shaped reward that gives a large penalty for an imminent collision with an enemy (Appendix G).

6.3 Extrapolating beyond the demonstrator's performance

Lastly, we tested whether the reward functions learned using D-REX can be used in conjunction with deep reinforcement learning to achieve better-than-demonstrator performance. We ran PPO on the learned reward function $\hat{R}_\theta$ for 1 million timesteps (MuJoCo tasks) and 50 million frames (Atari games). We ran three replicates of PPO with different seeds and report the best performance on the

Table 1: Comparison of the performance of D-REX with behavioral cloning (BC), GAIL [35], and the demonstrator's performance. Results are the best average ground-truth returns over 3 random seeds with 20 trials per seed. Bold denotes performance that is better than the best demonstration.

	Demonstrations		D-REX		BC		GAIL	
Task	Avg.	Best	Average	Stdev.	Avg.	Stdev.	Avg.	Stdev.
Hopper	1029.1	1167.9	**2072.0**	(1574.2)	943.8	(208.4)	**2700.2**	(692.3)
HalfCheetah	187.7	187.7	**972.9**	(96.1)	-115.9	(179.8)	85.2	(86.0)
Beam Rider	1,524.0	2,216.0	**7,220.0**	(2221.9)	1,268.6	(776.6)	1778.0	(787.1)
Breakout	34.5	59.0	**94.7**	(16.5)	29.75	(10.1)	0.3	(0.4)
Enduro	85.5	134.0	**247.9**	(88.4)	83.4	(27.0)	62.4	(24.0)
Pong	3.7	**14.0**	-9.5	(9.8)	8.6	(9.5)	-3.4	(3.8)
Q*bert	770.0	850.0	**22543.8**	(7434.1)	1,013.75	(721.1)	737.5	(311.4)
Seaquest	524.0	720.0	**801.0**	(4.4)	530.0	(109.8)	554.0	(108.8)
Space Invaders	538.5	930.0	**1,122.5**	(501.2)	426.5	(187.1)	364.8	(139.7)

ground-truth reward function, averaged over 20 trajectory rollouts. Table 1 compares the performance of the demonstrator with the performance of D-REX, behavioral cloning (BC), and Generative Adversarial Imitation Learning (GAIL) [35], a state-of-the-art imitation learning algorithm.

The results in Table 1 demonstrate that policies optimized using D-REX outperform the best demonstration in all tasks except for Pong. Furthermore, D-REX is also able to outperform BC and GAIL across all tasks except for Hopper and Pong. On the simulated MuJoCo robotics tasks, D-REX results in a 77% (Hopper) and 418% (HalfCheetah) performance increase when compared with the best demonstration. On Q*Bert, D-REX exploits a known loophole in the game which allows nearly infinite points. Excluding Q*Bert, D-REX results in an average performance increase of 39% across the Atari tasks, when compared with the best demonstration. To test the robustness of the policy learned via D-REX, we also considered the worst-case performance, something that is important for safe inverse reinforcement learning [36, 37]. As shown in Appendix C, D-REX achieves better worst-case performance than either the demonstrator or standard imitation learning algorithms. To ensure that D-REX is learning more than a simple bonus for staying alive, we also compared D-REX with a PPO agent trained with a +1 reward for every timestep. Our results in Appendix D demonstrate that D-REX is superior to a simple +1 reward across all games, except for Pong.

7 Conclusion

Imitation learning approaches are typically unable to outperform the demonstrator. This is because most approaches either directly mimic the demonstrator or find a reward function that makes the demonstrator appear near optimal. While algorithms that can exceed the performance of a demonstrator exist, they either rely on a significant number of active queries from a human [2, 5, 7], a hand-crafted reward function [1], or pre-ranked demonstrations [3]. Furthermore, prior research has lacked theory about when better-than-demonstrator performance is possible. We first addressed this lack of theory by presenting a sufficient condition for extrapolating beyond the performance of a demonstrator. We also provided theoretical results demonstrating how preferences and rankings allow for better reward function learning by reducing the learner's uncertainty over the true reward function.

We next focused on making reward learning from rankings more applicable to a wider variety of imitation learning tasks where only unlabeled demonstrations are available. We presented a novel imitation learning algorithm, Disturbance-based Reward Extrapolation (D-REX) that automatically generates ranked demonstrations via noise injection and uses these demonstrations to seek to extrapolate beyond the performance of a suboptimal demonstrator. We empirically evaluated D-REX on a set of simulated robot locomotion and Atari tasks and found that D-REX outperforms state-of-the-art imitation learning techniques and also outperforms the best demonstration in 8 out of 9 tasks. These results provide the first evidence that better-than-demonstrator imitation learning is possible without requiring extra information such as rewards, active supervision, or preference labels. Our results open the door to the application of a variety of ranking and preference-based learning techniques [31, 38] to standard imitation learning domains where only unlabeled demonstrations are available.

Acknowledgments

This work has taken place in the Personal Autonomous Robotics Lab (PeARL) at The University of Texas at Austin. PeARL research is supported in part by the NSF (IIS-1724157, IIS-1638107, IIS-1617639, IIS-1749204) and ONR(N00014-18-2243).

References

[1] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[3] D. Brown, W. Goo, N. Prabhat, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, 2019.

[4] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st international conference on Machine learning*, 2004.

[5] D. Sadigh, A. D. Dragan, S. S. Sastry, and S. A. Seshia. Active preference-based learning of reward functions. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[6] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei. Reward learning from human preferences and demonstrations in atari. In *Advances in Neural Information Processing Systems*, 2018.

[7] M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh. Learning reward functions by integrating human demonstrations and preferences. In *Proceedings of Robotics: Science and Systems (RSS)*, June 2019.

[8] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[9] S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877*, 2018.

[10] M. E. Taylor, H. B. Suay, and S. Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[11] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[12] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[13] D. H. Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3804–3809. IEEE, 2011.

[14] K. Shiarlis, J. Messias, and S. Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016.

[15] J. Zheng, S. Liu, and L. M. Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2198–2205, 2014.

[16] S. Choi, K. Lee, and S. Oh. Robust learning from demonstrations with mixed qualities using leveraged gaussian processes. *IEEE Transactions on Robotics*, 2019.

[17] U. Syed and R. E. Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2008.

[18] B. Eric, N. D. Freitas, and A. Ghosh. Active preference learning with discrete choice data. In *Advances in neural information processing systems*, pages 409–416, 2008.

[19] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[20] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning objective functions for manipulation. In *2013 IEEE International Conference on Robotics and Automation*, pages 1331–1336. IEEE, 2013.

[21] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008.

[22] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. *Conference on Robot Learning (CoRL)*, 2017.

[23] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[24] P. S. Castro, S. Li, and D. Zhang. Inverse reinforcement learning with multiple ranked experts. *arXiv preprint arXiv:1907.13411*, 2019.

[25] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM, 2012.

[26] C. Chuck, M. Laskey, S. Krishnan, R. Joshi, R. Fox, and K. Goldberg. Statistical data cleaning for deep learning of automation tasks from demonstrations. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1142–1149. IEEE, 2017.

[27] D. Kent, C. Saldanha, and S. Chernova. A comparison of remote robot teleoperation interfaces for general object manipulation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 371–379. ACM, 2017.

[28] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 663–670, 2000.

[29] S. Armstrong and S. Mindermann. Occam's razor is insufficient to infer the preferences of irrational agents. In *Advances in Neural Information Processing Systems*, pages 5598–5609, 2018.

[30] D. S. Brown and S. Niekum. Machine teaching for inverse reinforcement learning: Algorithms and applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7749–7758, 2019.

[31] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, 2007.

[32] R. D. Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2012.

[33] M. Bain and C. Sommut. A framework for behavioural claning. *Machine intelligence*, 15(15):103, 1999.

[34] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding atari agents. In *International Conference on Machine Learning*, pages 1787–1796, 2018.

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

[36] D. S. Brown and S. Niekum. Efficient Probabilistic Performance Bounds for Inverse Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[37] D. S. Brown, Y. Cui, and S. Niekum. Risk-aware active inverse reinforcement learning. *Conference on Robot Learning (CoRL)*, 2018.

[38] W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323, 2009.

[39] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.

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

A Theory and Proofs

We consider the case where the reward function of the demonstrator is approximated by a linear combination of features $R(s) = w^T\phi(s)$. Note that these can be arbitrarily complex features, such as the activations of the penultimate layer of a deep neural network. The expected return of a policy when evaluated on $R(s)$ is given by

$$J(\pi|R) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\right] = w^T\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)\right] = w^T\Phi_\pi, \tag{8}$$

where Φ_π are the expected discounted feature counts that result from following the policy π.

Theorem 1. *If the estimated reward function is $\hat{R}(s) = w^T\phi(s)$ and the true reward function is $R^*(s) = \hat{R}(s) + \epsilon(s)$ for some error function $\epsilon : \mathcal{S} \to \mathbb{R}$ and $\|w\|_1 \leq 1$, then extrapolation beyond the demonstrator, i.e., $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$, is guaranteed if:*

$$J(\pi^*_{R^*}|R^*) - J(\mathcal{D}|R^*) > \epsilon_\Phi + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{9}$$

where $\pi^*_{R^*}$ is the optimal policy under R^*, $\epsilon_\Phi = \|\Phi_{\pi^*} - \Phi_{\hat{\pi}}\|_\infty$ and $\|\epsilon\|_\infty = \sup\{\,|\epsilon(s)| : s \in \mathcal{S}\,\}$.

Proof. In order for extrapolation to be possible, the demonstrator must perform worse than $\hat{\pi}$, the policy learned via IRL, when evaluated under the true reward function. We define $\delta = J(\pi^*_{R^*}|R^*) - J(\mathcal{D}|R^*)$ as the optimality gap between the demonstrator and the optimal policy under the true reward function. We want to ensure that $J(\pi^*|R^*) - J(\hat{\pi}|R^*) < \delta$. We have

$$\begin{align}
J(\pi^*_{R^*}|R^*) - J(\hat{\pi}|R^*) &= \left|\mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)\big]\right| \tag{10}\\
&= \left|\mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t (w^T\phi(s_t) + \epsilon(s_t))\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t (w^T\phi(s_t) + \epsilon(s_t))\big]\right| \tag{11}\\
&= \left|w^T\mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)\big] + \mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big] - w^T\mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big]\right| \tag{12}\\
&= \left|w^T\Phi_{\pi^*} + \mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big] - w^T\Phi_{\hat{\pi}} - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big]\right| \tag{13}\\
&= \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big]\right| \tag{14}\\
&\leq \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \big[\sum_{t=0}^{\infty}\gamma^t\sup_{s\in\mathcal{S}}\epsilon(s)\big] - \big[\sum_{t=0}^{\infty}\gamma^t\inf_{s\in\mathcal{S}}\epsilon(s)\big]\right| \tag{15}\\
&= \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \big(\sup_{s\in\mathcal{S}}\epsilon(s) - \inf_{s\in\mathcal{S}}\epsilon(s)\big)\sum_{t=0}^{\infty}\gamma^t\right| \tag{16}\\
&\leq \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \frac{2\|\epsilon\|_\infty}{1-\gamma}\right| \tag{17}\\
&\leq |w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}})| + \left|\frac{2\|\epsilon\|_\infty}{1-\gamma}\right| \tag{18}\\
&\leq \|w\|_1\|\Phi_{\pi^*} - \Phi_{\hat{\pi}}\|_\infty + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{19}\\
&\leq \epsilon_\Phi + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{20}
\end{align}$$

where $\Phi_\pi = \mathbb{E}_\pi[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)]$ and line (19) results from Hölder's inequality. Thus, as long as $\delta > \epsilon_\Phi - \frac{2\|\epsilon\|_\infty}{1-\gamma}$, then $J(\pi^*|R^*) - J(\hat{\pi}|R^*) < J(\pi^*|R^*) - J(\mathcal{D}|R^*)$ and thus, $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$. □

Theorem 1 makes the assumption that the learner and demonstrator operate in the same state-space. However, because Theorem 1 only involve differences in expected features and reward errors over state visitations, the transition dynamics or action spaces are not required to be the same between the demonstrator and the learner. While our experiments use demonstrator actions to perform behavioral cloning, future work could use an imitation learning from observation method [39] to learn an initial cloned policy without requiring demonstrator actions. This would make D-REX robust to action differences between the demonstrator and learner and allow the learner to potentially learn a more efficient policy if its action space and transition dynamics are better suited to the task.

A.1 Extrapolation via ranked demonstrations

The previous results demonstrate that in order to extrapolate beyond a demonstrator, it is sufficient to have small reward approximation error. However, the following proposition, adapted with permission from [24], demonstrates that the reward functions inferred by an IRL or apprenticeship learning algorithm may be quite superficial and may not accurately represent some dimensions of the true reward function.

Proposition 1. *There exist MDPs with true reward function R^*, expert policy π_E, approximate reward function $\hat{R}$, and non-expert policies π_1 and π_2, such that*

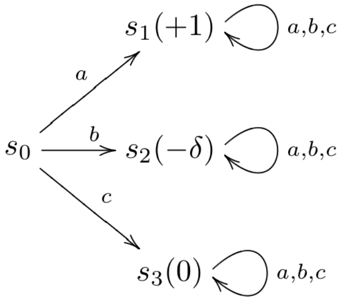
$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|R^*) \ \textit{ and } \ J(\pi_1|R^*) \ll J(\pi_2|R^*) \tag{21}$$

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \ \textit{ and } \ J(\pi_1|\hat{R}) = J(\pi_2|\hat{R}). \tag{22}$$

However, enforcing a preference ranking over trajectories, $\tau^* \succ \tau_2 \succ \tau_1$, where $\tau^* \sim \pi^*$, $\tau_2 \sim \pi_2$, and $\tau_1 \sim \pi_1$, results in a learned reward function $\hat{R}$, such that

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \ \textit{ and } \ J(\pi_1|\hat{R}) < J(\pi_2|\hat{R}). \tag{23}$$

Proof. Consider the MDP shown below. There are three actions a, b, c, with deterministic transitions.



Each transition is labeled by the action name. The true reward received upon entering a state is indicated in parenthesis, and $\delta \gg 0$ is some arbitrary constant. Clearly, $\pi_E(s_0) = a$. Setting $\hat{R}(s_1) = 1$, $\hat{R}(s_2) = \hat{R}(s_3) = 0$, $\pi_1(s_0) = b$, and $\pi_2(s_0) = c$ provides the existence proof for Equations (21) and (22).

Enforcing the preference constraints $\tau^* \succ \tau_1 \succ \tau_2$ for $\tau^* = (s_0, a, s_1)$, $\tau_2 = (s_0, b, s_2)$, $\tau_1 = (s_0, c, s_3)$, results in a learned reward function $\hat{R}$ such that $J(\tau^*|\hat{R}) > J(\pi_2|\hat{R}) \succ J(\pi_1|\hat{R})$ which finishes the proof. □

Proposition 1 gives a simple example of when learning from an expert demonstration reveals little about the underlying reward structure of the MDP. While it is true that solving the MDP in the above example only requires knowing that state s_1 is preferable to all other states, this will likely lead to an agent assuming that both s_2 and s_3 are equally undesirable.

This may be problematic for several reasons. The first problem is that learning a reward function from demonstrations is typically used as a way to generalize to new situations—if there is a change in the initial state or transition dynamics, an agent can still determine what actions it should take by transferring the learned reward function. However, if the learned reward function is drastically different than the true reward, this can lead to poor generalization, as would be the case if the dynamics in the above problem change and action a now causes a transition to state s_2. Another problem is that most learning from demonstration applications focus on providing *non-experts* the ability to program by example. Thus, the standard IRL approach of finding a reward function that maximizes the likelihood of the demonstrations may lead to reward functions that overfit to demonstrations and may be oblivious to important differences in rewards.

A natural way to alleviate these problems is via ranked demonstrations. Consider the problem of learning from a sequence of m demonstrated trajectories $\tau_1, \dots, \tau_m$, ranked according to preference such that $\tau_1 \prec \tau_2, \dots \prec \tau_m$. Using a set of strictly ranked demonstrations avoids the degenerate all-zero reward. Furthermore, ranked demonstrations provide explicit information about both what to do as well as what not to do in an environment.

A.2 Ranking Theory

IRL is an ill-posed problem due to reward ambiguity—given any policy, there are an infinite number of reward functions that make this policy optimal [11]. However, it is possible to analyze the amount of reward ambiguity. In particular, if the reward is represented by a linear combination of weights, then the feasible region of all reward functions that make a policy optimal can be defined as an intersection of half-planes [30]:

$$H_\pi = \bigcap_{\pi' \in \Pi} w^T(\Phi_\pi - \Phi_{\pi'}) \geq 0, \tag{24}$$

We define the *reward ambiguity*, $G(H_\pi)$, as the volume of this intersection of half-planes:

$$G(H_\pi) = \text{Volume}(H_\pi), \tag{25}$$

where we assume without loss of generality that $\|w\| \leq 1$, to ensure this volume is bounded.

We now prove that a total ranking over policies results in no more reward ambiguity than simply using the optimal policy.

Proposition 2. *Given a policy class Π, an optimal policy $\pi^* \in \Pi$ and a total ranking over Π, and a reward function $R(s) = w^T\phi(s)$, the reward ambiguity resulting from π^* is greater than or equal to the reward ambiguity of using a total ranking, i.e., $G(H^*_\pi) \geq G(H_{\text{ranked}})$.*

Proof. Consider policies π_1 and π_2 where $J_{R^*}(\pi_1) \geq J_{R^*}(\pi_2)$. We can write this return inequality in terms of half-spaces as follows:

$$\begin{aligned} & J_{R^*}(\pi_1) \geq J_{R^*}(\pi_2) && (26) \\ \iff\quad & w^T\Phi_{\pi_1} \geq w^T\Phi_{\pi_2} && (27) \\ \iff\quad & w^T(\Phi_{\pi_1} - \Phi_{\pi_2}) \geq 0 && (28) \end{aligned}$$

defining a half-space over weight vectors.

Consider the optimal policy $\pi^*_{r^*}$. This policy induces a set of half-space constraints over all other possible policies $\pi \in \Pi$. Thus we have the following half-space constraints:

$$H_{\pi^*} = \bigcap_{\pi \in \Pi} w^T(\Phi_{\pi^*} - \Phi_\pi) \geq 0 \tag{29}$$

However, if we have a total ordering over Π, then we have the following intersection of half-spaces

$$\begin{aligned} H_{\text{ranked}} &= \bigcap_{\pi_i \succsim \pi_j \in \Pi} w^T(\Phi_{\pi_i} - \Phi_{\pi_j}) \geq 0 && (30) \\ &= H_{\pi^*} \cap \Bigg(\bigcap_{\substack{\pi_i \succsim \pi_j \in \Pi \\ \pi_i \neq \pi^*}} w^T(\Phi_{\pi_i} - \Phi_{\pi_j}) \geq 0 \Bigg). && (31) \end{aligned}$$

Thus, $H_{\text{ranked}} \subseteq H_{\pi^*}$, and the volume of the set of feasible reward functions induced by the total ranking is therefore less than or equal to the volume of H_{π^*}, i.e., $G(H_{\text{ranked}}) \leq G(H_{\pi^*})$.

□

A.3 Uncertainty Reduction for Random Halfspaces

In this section we seek to bound how many reward hypotheses are removed after a certain number of ranked trajectories, $\tau_i \prec \tau_j$. Our main result that we will prove is that sampling random half-space constraints causes the reward ambiguity to decrease exponentially.

We assume that the true reward function, $R^*(s) = {w^*}^T\phi(s)$, is a linear combination of features $w \in \mathbb{R}^n$. Let $\mathcal{H}$ denote the set of all reward hypotheses. We assume a linear combination of features $R(s) = w^T\phi(s)$, so $\mathcal{H} = \mathbb{R}^n$. We also make the common assumption that $\|w\|_1 \leq 1$ so that $\mathcal{H}$ is bounded [4, 36].

Each pair of trajectories, $\tau_i \prec \tau_j$, that are ranked based on R^*, forms a half-space constraint over the true reward feature weights ${w^*}^T$:

$$\begin{aligned} & \tau_i \prec \tau_j && (32)\\ \Rightarrow \quad & J(\tau_i|R^*) < J(\tau_j|R^*) && (33)\\ \Rightarrow \quad & {w^*}^T\Phi_{\tau_i} < {w^*}^T\Phi_{\tau_j} && (34)\\ \Rightarrow \quad & {w^*}^T(\Phi_{\tau_j} - \Phi_{\tau_i}) > 0, && (35) \end{aligned}$$

where $\Phi_\tau = \sum_{t=0}^T \gamma^t\phi(s_t)$ for $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$.

We define Ξ to be the set of all trajectories in the MDP. Let $\mathcal{X}$ be the set of half-space constraints that result from all ground-truth rankings over all possible trajectory pairs from Ξ, i.e.,

$$\mathcal{X} = \big\{(\Phi_{\tau_j} - \Phi_{\tau_i}) : J(\tau_j|R^*) > J(\tau_j|R^*), \tau_i, \tau_j \in \Xi\big\}. \tag{36}$$

We define the space of all reward feature weight vectors that are consistent with $\mathcal{X}$ as $\mathcal{R}$, where

$$\mathcal{R} = \bigg\{w : w \in \bigcap_{x\in\mathcal{X}} w^Tx > 0\bigg\}, \tag{37}$$

where x is one of the half-space normal vectors from the set $\mathcal{X}$. We also define $\mathcal{J} = \mathcal{H} \setminus \mathcal{R}$, i.e., the space that gets cut by all the half-spaces defined by the normal vectors in $\mathcal{X}$. Our goal is to find a lower bound, in expectation, on how much volume is cut from $\mathcal{J}$ for k half-space constraints.

To simplify our proofs and notation, we assume there are only a finite number of reward weight hypotheses in $\mathcal{J}$. Denote j as the jth reward hypothesis in $\mathcal{J}$ and let $J = |\mathcal{J}|$ be the total number of these hypotheses. For each halfspace $x \in \mathcal{X}$, define $Z_i(x)$ as the binary indicator of whether halfspace x (assumed to be represented by its normal vector) eliminates the hypothesis $j \in \mathbb{R}^n$, i.e.

$$Z_j(x) = \begin{cases} 1 & \text{if } j^Tx \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{38}$$

We define

$$\mathcal{T} = \{j^{(\sum_{x\in\mathcal{X}} Z_j(x))} : j \in \mathcal{J}\} \tag{39}$$

to be the multiset containing all the hypotheses in $\mathcal{J}$ that are eliminated by the half-space constraints in $\mathcal{X}$. This is a multiset since we include repeats of j that are eliminated by different half-spaces in $\mathcal{X}$. We use the notation $\{a^{(y)}\}$ to denote the multiset with y copies of a. We define $T = |\mathcal{T}|$ to be the sum of the total number of reward hypotheses that are eliminated by each half-space in $\mathcal{X}$. Similarly, we define $H = |\mathcal{H}|$ and $X = |\mathcal{X}|$ to be the number of hypothesis in $\mathcal{H}$ and the number of half-spaces in $\mathcal{X}$, respectively.

Our goal is to find a lower bound on the expected number of hypotheses that are removed given k half-space constraints that result from k pairwise trajectory preferences.

Note that we can define T as

$$T = \sum_{j\in J}\sum_{x\in X} Z_j(x). \tag{40}$$

We can derive the expected number of unique hypotheses in $\mathcal{J}$ that are eliminated by the first half-space as follows:

$$\begin{aligned}
\mathbb{E}[\sum_{j\in\mathcal{J}} Z_j(x)|x\sim\mathcal{U}(\mathcal{X})] &= \sum_{j\in\mathcal{J}} \mathbb{E}_{x\in\mathcal{X}}[Z_j(x)] && (41)\\
&= \sum_{j\in\mathcal{J}}\sum_{x\in X} p(x)Z_j(x) && (42)\\
&= \sum_{j\in\mathcal{J}}\sum_{x\in\mathcal{X}} \frac{1}{X}Z_j(x) && (43)\\
&= \frac{1}{X}\sum_{j\in\mathcal{J}}\sum_{x\in\mathcal{X}} Z_j(x) && (44)\\
&= \frac{T}{X} && (45)
\end{aligned}$$

where $\mathcal{U}(\mathcal{X})$ represents a uniform distribution over $\mathcal{X}$. There are T hypotheses left to eliminate and X halfspaces left to choose from. Once we choose all of them we have eliminated all of $\mathcal{T}$, so, in expectation, each half-space elminates T/X of the hypotheses in $\mathcal{T}$. We can now consider T to be the total number of hypotheses that remain to be eliminated after already enforcing some number of half-space constraints, and the same reasoning applies. We will take advantage of this later to form a recurrence relation over the elements left in $\mathcal{J}$, i.e. the reward function hypotheses that are false, but have not yet been eliminated by a pairwise trajectory ranking.

When selecting a half-space from $\mathcal{X}$ uniformly, we eliminate T/X unique hypotheses in $\mathcal{J}$ in expectation. If we assume that each $j\in\mathcal{J}$ is distributed uniformly across the half-spaces, then on average, there are T/J copies of a hypothesis j in $\mathcal{T}$. This can be shown formally as follows:

$$\begin{aligned}
\mathbb{E}_{j\in\mathcal{J}}[\sum_{x\in\mathcal{X}} Z_j(x)] &= \sum_{j\in\mathcal{J}} p(j)\Big[\sum_{x\in\mathcal{X}} Z_j(x)\Big] && (46)\\
&= \frac{1}{J}\sum_{j\in\mathcal{J}}\sum_{x\in\mathcal{X}} Z_j(x) && (47)\\
&= \frac{T}{J}. && (48)
\end{aligned}$$

Thus, selecting a half-space uniformly at random eliminates T/X unique hypotheses from $\mathcal{J}$; however, there are T/J copies of each hypothesis on average, so a random half-space eliminates $\frac{T^2}{JX}$ total hypotheses from $\mathcal{T}$ on average.

Recurrence relation The above analysis results in the following system of recurrence relations where J_k is the number of unique hypotheses that have not yet been eliminated after k half-spaces constraints, T_k is the sum of the total hypotheses (including repeats) that have not yet been eliminated the first k half-spaces, and X is the total number of half-space constraints.

$$\begin{aligned}
&J_0 = J && (49)\\
&T_0 = T && (50)\\
&J_{k+1} = J_k - \frac{T_k}{X-k} && (51)\\
&T_{k+1} = T_k - \frac{T_k^2}{J_k\cdot(X-k)} && (52)
\end{aligned}$$

To simplify the analysis we make the following assumptions. First we assume that $\mathcal{R}$ is small compared to the full space of $\mathcal{H}$. Thus, $\mathcal{H}\approx\mathcal{J}$. Since each $x\in\mathcal{X}$ covers $H/2$ hypotheses, then each $x\in\mathcal{X}$ covers approximately $J/2$ hypotheses. Summing over all halfspaces, we have that $T_0\approx\sum_{x\in\mathcal{X}} J/2 = XJ/2$. Furthermore, if X, the number of possible half-spaces, is large, then $X-k\approx X$.

Given these assumptions we now have the following simplified system of recurrence relations:

$$J_0 = J \tag{53}$$

$$T_0 = \frac{J_0 X}{2} \tag{54}$$

$$J_{k+1} = J_k - \frac{T_k}{X} \tag{55}$$

$$T_{k+1} = T_k - \frac{T_k^2}{J_k X} \tag{56}$$

where T_k and J_k are the number of elements in $\mathcal{T}$ and $\mathcal{J}$ after k half-spaces constraints have been applied.

The solution to these recurrences is:

$$J_k = \frac{J_0}{2^k}, \tag{57}$$

$$T_k = \frac{J_0 X}{2^{k+1}}. \tag{58}$$

and can be verified by plugging the solution into the above equations and checking that it satisfies the base cases and recurrence relations.

We can now bound the number of half-space constraints (the number of ranked random trajectory pairs) that are needed to get $J = |\mathcal{J}|$ down to some level ϵ.

Theorem 2. *To reduce the volume of $\mathcal{J}$ such that $J_k = \epsilon$, then it suffices to have k random half-space constraints, where*

$$k = \log_2 \frac{J_0}{\epsilon} \tag{59}$$

Proof. As shown above, we have that

$$J_k = \frac{J_0}{2^k}. \tag{60}$$

Solving for k and substituting ϵ for J_k we get

$$k = \log_2 \frac{J_0}{\epsilon} \tag{61}$$

□

This means that J_k decreases exponentially with k, meaning we only need to sample a logarithmic number of half-space constraints to remove a large portion of the possible reward hypotheses.

Corollary 1. *To reduce J by x% it suffices to have*

$$k = \log_2 \left(\frac{1}{1 - x/100} \right) \tag{62}$$

random half-space constraints.

Proof. We want

$$\frac{x}{100} = \frac{J_0 - \frac{J_0}{2^k}}{J_0} = 1 - \frac{1}{2^k} \tag{63}$$

Solving for k results in

$$k = \log_2 \left(\frac{1}{1 - x/100} \right). \tag{64}$$

□

B Noise Degradation

The full set of noise degradation plots for all seven Atari games are shown in Figure 4. For MuJoCo experiments, we used 20 different noise levels evenly spaced over the interval $[0.0, 1.0)$ and generated 5 trajectories for each level. For the Atari experiments, we used the noise schedule $\mathcal{E} = (0.01, 0.25, 0.5, 0.75, 1.0)$ and generated $K = 20$ trajectories for each level.

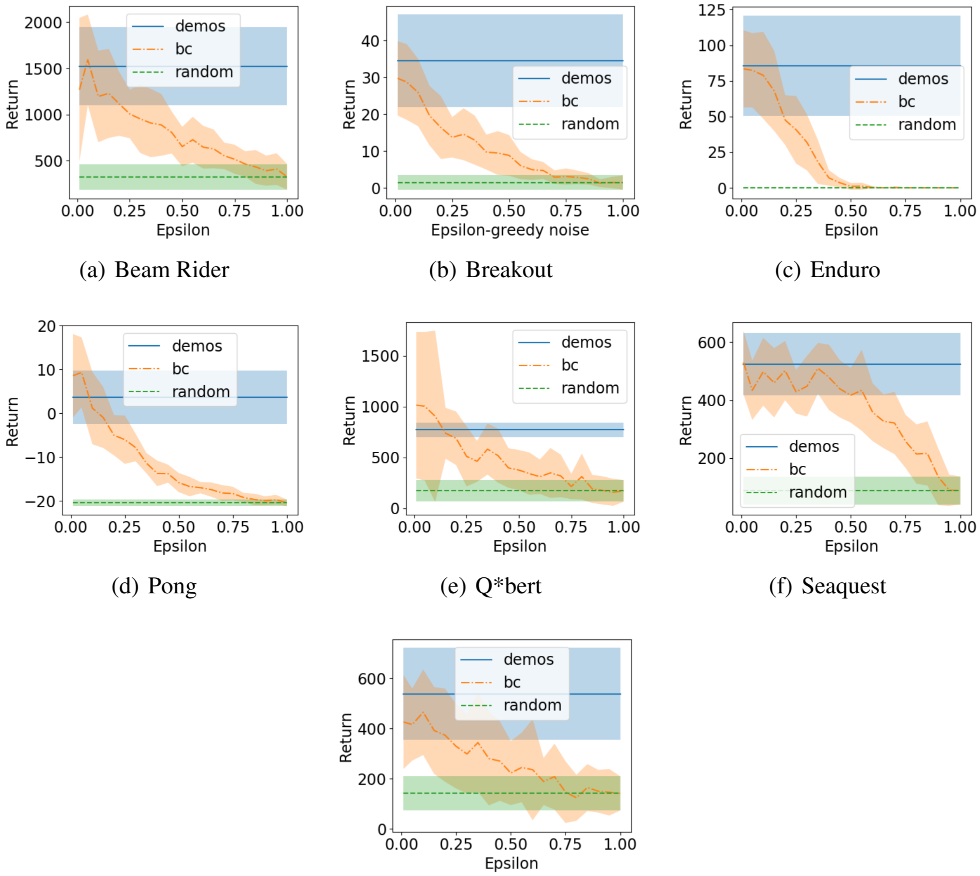
For the MuJoCo tasks, we used the following epsilon greedy policy:

$$\pi_{\rm BC}(s_t|\epsilon) = \begin{cases} \pi_{\rm BC}(s_t), & \text{with probability } 1-\epsilon \\ a_t \sim \mathcal{U}([-1,1]^n), & \text{with probability } \epsilon. \end{cases} \tag{65}$$

For Atari, we used the following epsilon greedy policy:

$$\pi_{\rm BC}(a_t|s_t,\epsilon) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|\mathcal{A}|}, & \text{if } \pi_{\rm BC}(s_t) = a_t \\ \frac{\epsilon}{|\mathcal{A}|}, & \text{otherwise} \end{cases} \tag{66}$$

where $|\mathcal{A}|$ is the number of valid discrete actions in each environment.



(a) Beam Rider

(b) Breakout

(c) Enduro

(d) Pong

(e) Q*bert

(f) Seaquest

(g) Space Invaders

Figure 4: The performance degradation of an imitation policy learned via behavioral cloning as the probability of taking a random action increases. Behavioral cloning is done on 10 demonstrations. Plots show mean and standard deviations over 20 rollouts per noise level.

B.1 Noise Injection Theory

Let ϵ be the probability of taking a random action. We assume that $\mathcal{A}$ is finite, and assume a finite horizon MDP with a horizon of T.

Optimal policy We start by analyzing what happens when we inject noise into an optimal policy, π^*. We will inject noise using the ϵ-greedy policy defined in Equation (66). We define p_ϵ to be the probability of taking an suboptimal action. For simplicity, we assume that there is only one optimal action at each state and that the states visited are independent of each other. This simplifies the analysis, but ignores the problems of compounding error that occur in sequential decision making tasks [40]. We will address the impact of compounding errors below.

Given the above assumptions we have

$$p_\epsilon = \frac{\epsilon(|A|-1)}{|A|}, \tag{67}$$

where A is the set of actions in the MDP.

If we define X to be the random variable representing the number of suboptimal actions in a trajectory of length T, then we can then analyze the number of suboptimal actions in the epsilon greedy policy $\pi^*(\cdot|\epsilon)$ using a binomial distribution with T trials and probability $p = p_\epsilon$.

This gives us $\mathbb{E}[X] = Tp_\epsilon$. If we assume that $|A|$ is large, then $p_\epsilon \approx \epsilon$ and we have $\mathbb{E}[X] = T\epsilon$. Thus, as ϵ goes from 0 to 1, the expected number of suboptimal actions in the policy interpolates between 0 and T. Furthermore, a standard Hoeffding bound shows that for large T, i.e., long trajectories, the empirical number of suboptimal actions will concentrate tightly around the mean $T\epsilon$. Thus, as ϵ goes from 0 to 1, with high probability the empirical number of suboptimal actions will interpolate between 0 and T.

Suboptimal cloned policy Suppose that instead of injecting noise into the optimal policy, we now inject noise into a policy, π_{BC}, that is cloned from suboptimal demonstrations. Define β to be the probability that π_{BC} takes the optimal action in a state. Note that here we also simplify the analysis by assuming that β is a constant that does not depend on the current state. We define p_{BC}^{opt} to be the probability of taking a suboptimal action in state s if π_{BC} would take the optimal action in that state. Similarly, we define p_{BC}^{sub} to be the probability of taking a suboptimal action in state s if π_{BC} would take a suboptimal action in that state.

As before we define p_ϵ to be the probability that $\pi_{\mathrm{BC}}(\cdot|\epsilon)$ takes a suboptimal action. Thus, we have that

$$p_\epsilon = \beta p_{\mathrm{BC}}^{opt} + (1-\beta)p_{\mathrm{BC}}^{sub}, \tag{68}$$

where

$$p_{\mathrm{BC}}^{opt} = \frac{\epsilon(|A|-1)}{|A|} \tag{69}$$

and

$$p_{\mathrm{BC}}^{sub} = 1 - \epsilon + \frac{\epsilon(|A|-1)}{|A|}. \tag{70}$$

Thus, we have

$$\begin{aligned} p_\epsilon &= \beta \cdot \frac{\epsilon(|A|-1)}{|A|} + (1-\beta)\cdot(1-\epsilon+\frac{\epsilon(|A|-1)}{|A|}) & (71)\\ &= (1-\beta)(1-\epsilon) + \frac{\epsilon(|A|-1)}{|A|}. & (72)\end{aligned}$$

As expected, when $\beta = 1$, then the first term is zero and we have the same result as for an optimal policy. Also, if $\epsilon = 1$, then all actions are random and we get the same result as for an optimal policy, since the optimality of the cloned policy does not affect the analysis if all actions are random.

If we assume that $|A|$ is large, then we have $(|A|-1)/|A| \approx 1$ and we have

$$p_\epsilon = 1 - \beta(1-\epsilon). \tag{73}$$

Thus, the expected number of suboptimal actions in the epsilon greedy cloned policy is

$$\mathbb{E}[X] = T \cdot (1 - \beta(1-\epsilon)). \tag{74}$$

With no noise injection ($\epsilon = 0$), we expect the cloned policy to make $(1-\beta)T$ suboptimal actions. Thus, for large enough A and T, as ϵ goes from 0 to 1 the empirical number of suboptimal actions will, with high probability, interpolate between $(1-\beta)T$ and T.

Table 2: Comparison of the average and worst-case performance of D-REX with respect to the demonstrator. Results are the worst-case performance corresponding to the results shown in Table 1 in the main text. Bold denotes worst-case performance that is better than the worst-case demonstration.

Worst-Case Performance				
Game	Demonstrator	D-REX	BC	GAIL
Beam Rider	900	**2916**	528	352
Breakout	17	**62**	13	0
Enduro	37	**152**	35	13
Pong	-5	-21	**-2**	-14
Q*bert	575	**7675**	650	350
Seaquest	320	**800**	280	260
Space Invaders	190	**575**	120	235

Compounding errors The above analysis has ignored the compounding errors that are accumulated when running a behavioral cloned policy in a sequential decision making task. Using the same analysis as Ross et al. [40] we get the following result that proves that the performance of the cloned policy with noise injection ϵ has an increasingly large performance gap from the expert as ϵ goes from 0 to 1.

Corollary 2. *Given a cloned policy π_{BC} from expert demonstrations, if the cloned policy makes a mistake with probability $1-\beta$, then*

$$J(\pi_{\mathrm{BC}}(\cdot|\epsilon)) \leq J(\pi^*) + T^2(1-\beta(1-\epsilon)) \tag{75}$$

where T is the time-horizon of the MDP.

Proof. As demonstrated above, given $|A|$ sufficiently large, we can define p_ϵ, the probability of taking a suboptimal action when following $\pi_{BC}(\cdot|\epsilon)$ is as follows:

$$\begin{aligned} p_\epsilon &= \beta\frac{\epsilon(|A|-1)}{|A|} + (1-\beta)(1-\epsilon+\frac{\epsilon(|A|-1)}{|A|}) & (76)\\ &= (1-\beta)(1-\epsilon) + \frac{\epsilon(|A|-1)}{|A|} & (77)\\ &\approx 1-\beta(1-\epsilon). & (78)\end{aligned}$$

The inequality then follows from the proof of Theorem 2.1 in [40]. □

C Risk Analysis

Addressing the worst-case performance is important for safe imitation learning [36, 37]. We investigated the worst-case performance of the demonstrator as it compares to the worst-case performance of the policy learned using D-REX. The results are shown in Table 2. Our results show that D-REX is able to learn safer policies than the demonstrator on 6 out of 7 games via intent extrapolation. The results show that on all games, except for Pong, D-REX is able to find a policy with both higher expected utility (see Table 1 in the main text) as well as higher worst-case utility (Table 2) when compared to the worst case performance of the demonstrator. D-REX also has a better worst-case performance than BC and GAIL across all games except for Pong.

D Live-Long Baseline

Here we describe a simple live-long baseline experiment to test that D-REX is actually learning something other than a positive bonus for living longer. Because we normalize the D-REX learned reward using a sigmoid, one concern is that the non-negativity of the sigmoid is the only thing that is needed to perform well on the Atari domain since games typically involve trying to stay alive as long as possible. We tested this by creating a live-long baseline that always rewards the agent with a +1 reward for every timestep. Table 3 shows that while a +1 reward is sufficient to achieve a moderate score on some games, it is insufficient to learn to play the games Enduro and Seaquest, both of which

Table 3: Comparison of D-REX with other imitation learning approaches. BC is behavioral cloning. Live-long assigns every observation a +1 reward and is run using an experimental setup identical to D-REX.

	Demonstrator		Imitation method		
Game	Avg.	Best.	D-REX	Live-Long	BC
Beam Rider	1524	2216	**7220**	5583.5	1268.6
Breakout	34.5	59	**94.7**	68.85	29.75
Enduro	85.5	134	**247.9**	0	83.4
Pong	3.7	**14**	-9.5	-5.3	8.6
Q*bert	770	850	**22543.8**	17073.75	1013.75
Seaquest	524	720	**801**	1	530
Space Invaders	538.5	930	**1122.5**	624	426.5

D-REX is able to learn to play. The reason that a +1 reward does not work on Enduro and Seaquest is that, in these games, it is possible to do nothing and cause an arbitrarily long episode. Thus, simply rewarding long episodes is not sufficient to learn to actually play. While a +1 reward was sufficient to achieve moderate to good scores on the other games, live-long reward is only able to surpass the performance of D-REX on Pong, which gives evidence that even if games where longer trajectories are highly correlated with the ground-truth return, D-REX is not simply rewarding longer episodes, but also rewarding trajectories that follow the demonstrators intention. This is also backed up by our reward attention heat maps shown later, which demonstrate that D-REX is paying attention to details in the observations which are correlated with the ground-truth reward.

E D-REX Details

Code and videos can be found at our project website https://dsbrown1331.github.io/CoRL2019-DREX/.

E.1 Demonstrations

To create the demonstrations, we used a partially trained Proximal Policy Optimization (PPO) agent that was checkpointed every 5 optimization steps (corresponds to 10,240 simulation steps) for MuJoCo experiments and 50 optimization steps (corresponds to 51,200 simulation steps) for Atari experiments. To simulate suboptimal demonstrations, we selected demonstration checkpoints such that they resulted in an average performance that was significantly better than random play, but also significantly lower than the maximum performance achieved by PPO when trained to convergence on the ground-truth reward. All checkpoints are included in the source code included in the supplemental materials.

E.2 Behavioral cloning

MuJoCo experiments We generated a trajectory of length 1,000, and the given 1,000 pairs of data is used for training. The policy network is optimized with L_2 loss for 10,000 iterations using Adam optimizer with a learning rate of 0.001 and a minibatch size of 128. Weight decay regularization is also applied in addition to regular loss term with a coefficient of 0.001. A multi-layer perceptron (MLP) having 4 layers and 256 units in the middle is used to parameterize a policy.

Atari experiments We used the state-action pairs from the 10 demonstrations and partitioned them into an 80% train 20% validation split. We used the Nature DQN network architecture and trained the imitation policy using Adam with a learning rate of 0.0001 and a minibatch size of 32. The state consists of four stacked frames which are normalized to have a value between 0 and 1, and the scores in the game scene are masked as it is done in Brown et al. [3]. We used the validation set for early stopping. In particular, after every 1000 updates on the training data we fully calculated the validation error of the current model. We trained the imitation policy until the validation loss failed to improve for 6 consecutive calculations of the validation error.

E.3 Synthetic rankings

We then used the cloned policy and generated 100 synthetic demonstrations for different noise levels. For the MuJoCo experiments, we used 20 different noise levels evenly spaced over the interval $[0.0, 1.0)$ and generated 5 trajectories for each level.

For the Atari experiments, we used the noise schedule $\mathcal{E} = (1.0, 0.75, 0.5, 0.25, 0.02)$ and generated $K = 20$ trajectories for each level. We found that a non-zero noise was necessary for most Atari games since deterministic policies learned through behavior cloning will often get stuck in a game and fail to take an action to continue playing. For example, in Breakout, it is necessary to release the ball after it falls past the paddle, and a deterministic policy may fail to fire a new ball. For a similar reason, we also found it beneficial to include a few examples of no-op trajectories to encourage the agent to actually complete the game. For each game, we created an additional “no-op" demonstration set comprised of four length 500 no-op demonstrations. Without these no-op demonstrations, we found that often the learned reward function would give a small positive reward to the agent for just staying alive and sometimes the RL algorithm would decide to just sit at the start screen and accumulate a nearly indefinite stream of small rewards rather than play the game. Adding no-op demonstrations as the least preferred demonstrations shapes the reward function such that it encourages action and progress. While this does encode some amount of domain knowledge into the reward function, it is common that doing nothing is worse than actually attempting to complete a task. We note that in extremely risky scenarios, it may be the case that always taking the no-op action is optimal, but leave these types of domains for future work.

E.4 Reward function training

For reward function training, we generally followed the setup used in [3]. We build a dataset of paired trajectory snippets with ranking, first by choosing two trajectories from given demonstrations and synthetic demonstrations, then by subsampling a snippet from each of trajectory.

MuJoCo experiments We built 3 datasets having different 5,000 pairs and trained a reward function for each of dataset using a neural network. Then, the ensemble of three neural network was used for reinforcement learning step. When two trajectories are selected from synthetic demonstration set to build a dataset, we discarded a pair whose epsilon difference is smaller than 0.3. This stabilizes a reward learning process by eliminating negative samples. Also, when subsampling from a whole trajectory, we limited the maximum length of snippet as 50 while there is no limitation on the minimum length. We then trained each neural network for 1,000 interactions with Adam optimizer with a learning rate of 1e-4 and minibatch size of 64. Weight decay regularization is also used with a coefficient of 0.01. A 3-layer MLP with 256 units in the middle is used to parameterize a reward function.

Atari experiments To generate training samples, we performed data augmentation to generate 40,000 training pairs. We first sampled two noise levels ϵ_i and ϵ_j, we then randomly sampled one trajectory from each noise level. Finally, we randomly cropped each trajectory keeping between 50 and 200 frames. Following the advice in [3], we also enforced a progress constraint such that the randomly cropped snippet from the trajectory with lower noise started at an observation timestep no earlier than the start of the snippet from the higher noise level. To speed up learning, we also only kept every 4th observation. Because observations are stacks of four frames this only removes redundant information from the trajectory. We assigned each trajectory pair a label indicating which trajectory had the lowest noise level.

Given our 40,000 labeled trajectory pairs, we optimized the reward function $\hat{R}_\theta$ using Adam with a learning rate of 1e-5. We held out 20% of the data as a validation set and optimized the reward function on the training data using the validation data for early stopping. In particular, after every 1000 updates we fully calculated the validation error of the current model. We stopped training once the validation error failed to improve for 6 consecutive calculations of the validation error.

We used an architecture having four convolutional layers with sizes 7x7, 5x5, 3x3, and 3x3, with strides 3, 2, 1, and 1. The 7x7 convolutional layer used 32 filters and each subsequent convolutional layer used 16 filters and LeakyReLU non-linearities. We then used a fully connected layer with 64 hidden units and a single scalar output. We fed in stacks of 4 frames with pixel values normalized between 0 and 1 and masked reward-related information from the scene; the game score and number of lives, the sector number and number of enemy ships left on Beam Rider, the bottom half of the dashboard for Enduro to mask the position of the car in the race, the number of divers found and the oxygen meter for Seaquest, and the power level and inventory for Hero.

E.5 Policy optimization

We optimized a policy by training a PPO agent on the learned reward function. We used the default hyperparameters in OpenAI Baselines.[3] Due to the variability that results from function approximation when using PPO, we trained models using seeds 0, 1, and 2 and reported the best results among them.

MuJoCo experiments We trained an agent for 1 million steps, and gradient is estimated for every 4,096 simulation steps. As same as the original OpenAI implementation, we normalized a reward with running mean and standard deviation. Model ensemble of three neural network is done by averaging such normalized reward.

Atari experiments 9 parallel workers are used to collect trajectories for policy gradient estimation. To reduce reward scaling issues, we followed the procedure proposed by Brown et al. [3] and normalized predicted rewards by feeding the output of $\hat{R}_\theta(s)$ through a sigmoid function before passing it to PPO. We trained PPO on the learned reward function for 50 million frames to obtain our final policy.

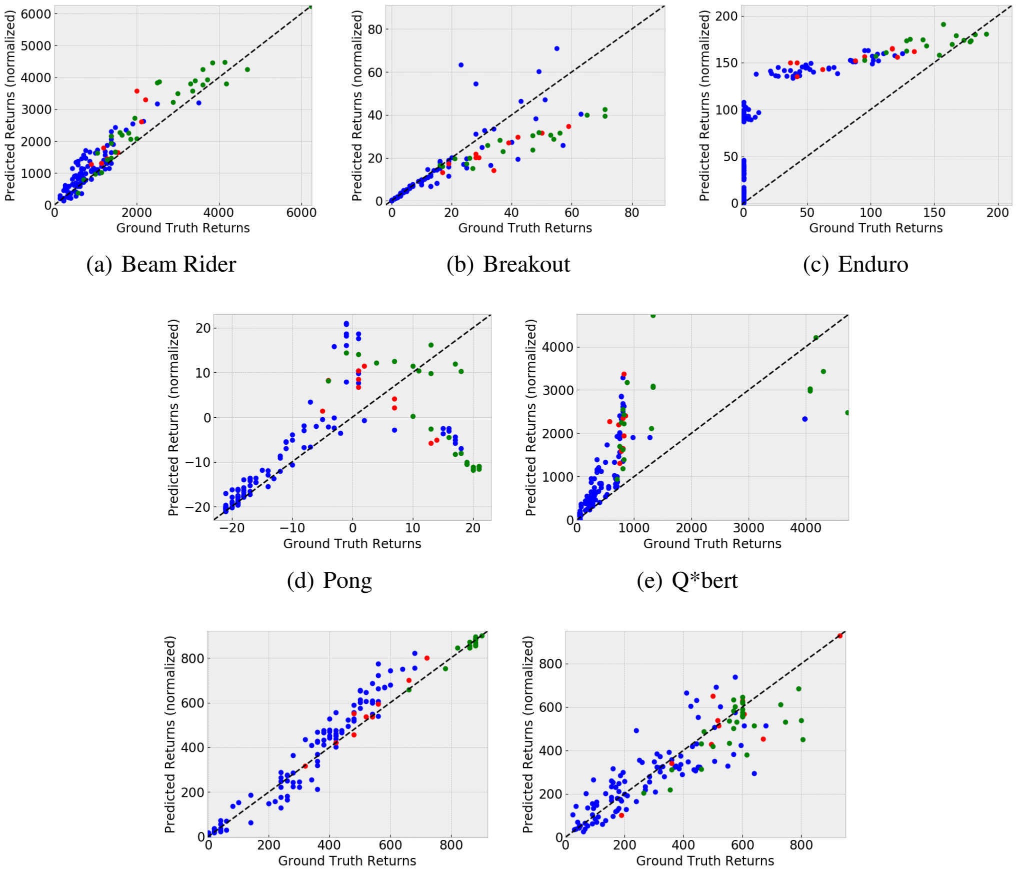
F GAIL

We used the default implementation of GAIL from OpenAI Baselines for Mujoco. For Atari we made a few changes to get the Baselines implementation to work with raw pixel observations. For the generator policy we used the Nature DQN architecture. The discriminator takes in a state (stack of four frames) and action (represented as a 2-d one-hot vector of shape (84,84,$|\mathcal{A}|$) that is concatenated to the 84x84x4 observation). The architecture for the discriminator is the same as the generator, except that it only outputs two logit values for discriminating between the demonstrations and the generator. We performed one generator update for every discriminator update.

G Reward Extrapolation and Attention Heatmaps

Figure 5 shows the reward extrapolation plots for all seven Atari games. Figures 6–12 show reward heatmaps for all seven Atari games. We generated the heatmaps using the technique proposed in [3]. We take a 3x3 mask and run it over every frame in an observation and compute the difference in predicted reward before and after the mask is applied. We then use the cumulative sum over all masks for each pixel to plot the heatmaps.

[3] https://github.com/openai/baselines



(a) Beam Rider

(b) Breakout

(c) Enduro

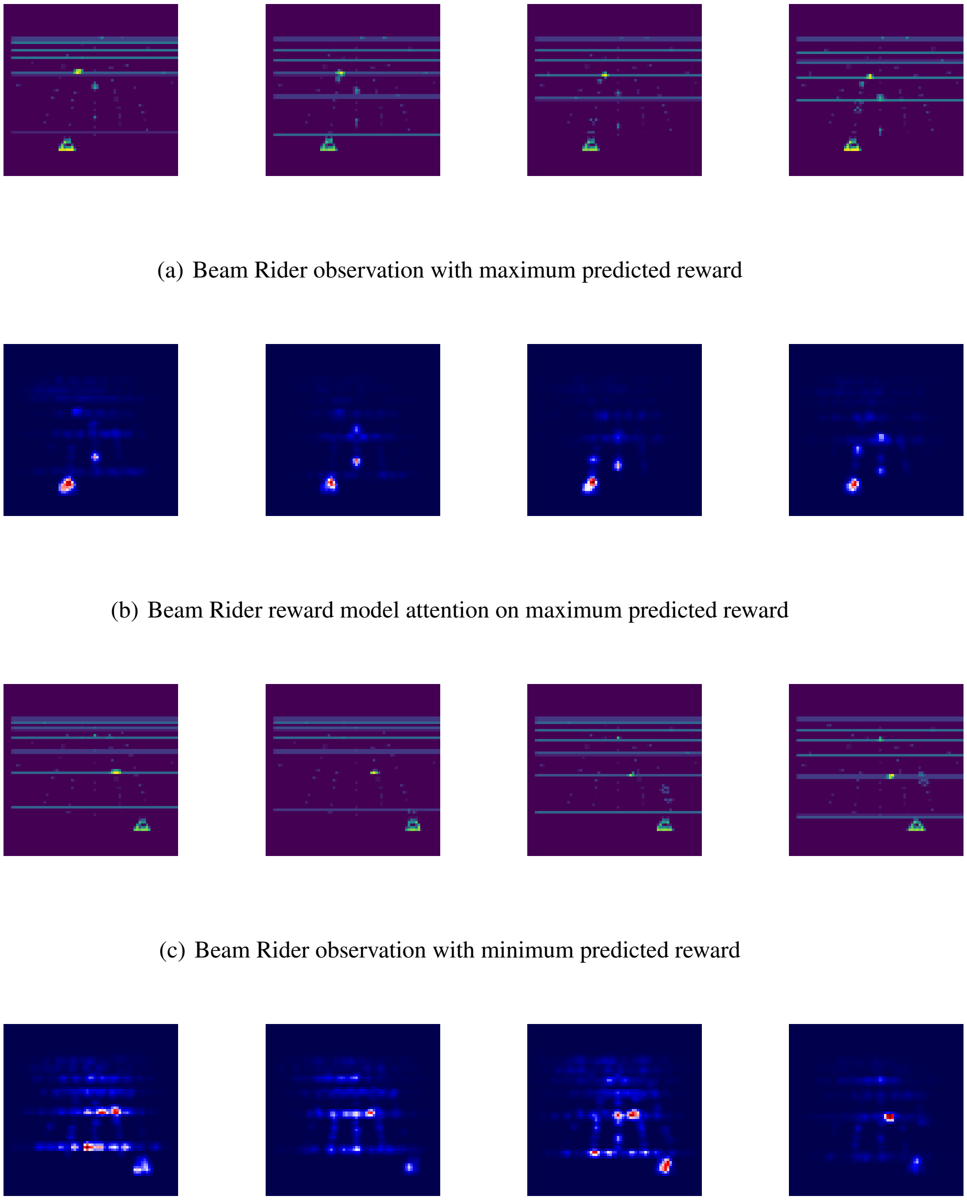
(d) Pong

(e) Q*bert

(f) Seaquest

(g) Space Invaders

Figure 5: Extrapolation plots for Atari games. Blue dots represent synthetic demonstrations generated via behavioral cloning with different amounts of noise injection. Red dots represent actual demonstrations, and green dots represent additional trajectories not seen during training. We compare ground truth returns over demonstrations to the predicted returns using D-REX (normalized to be in the same range as the ground truth returns).



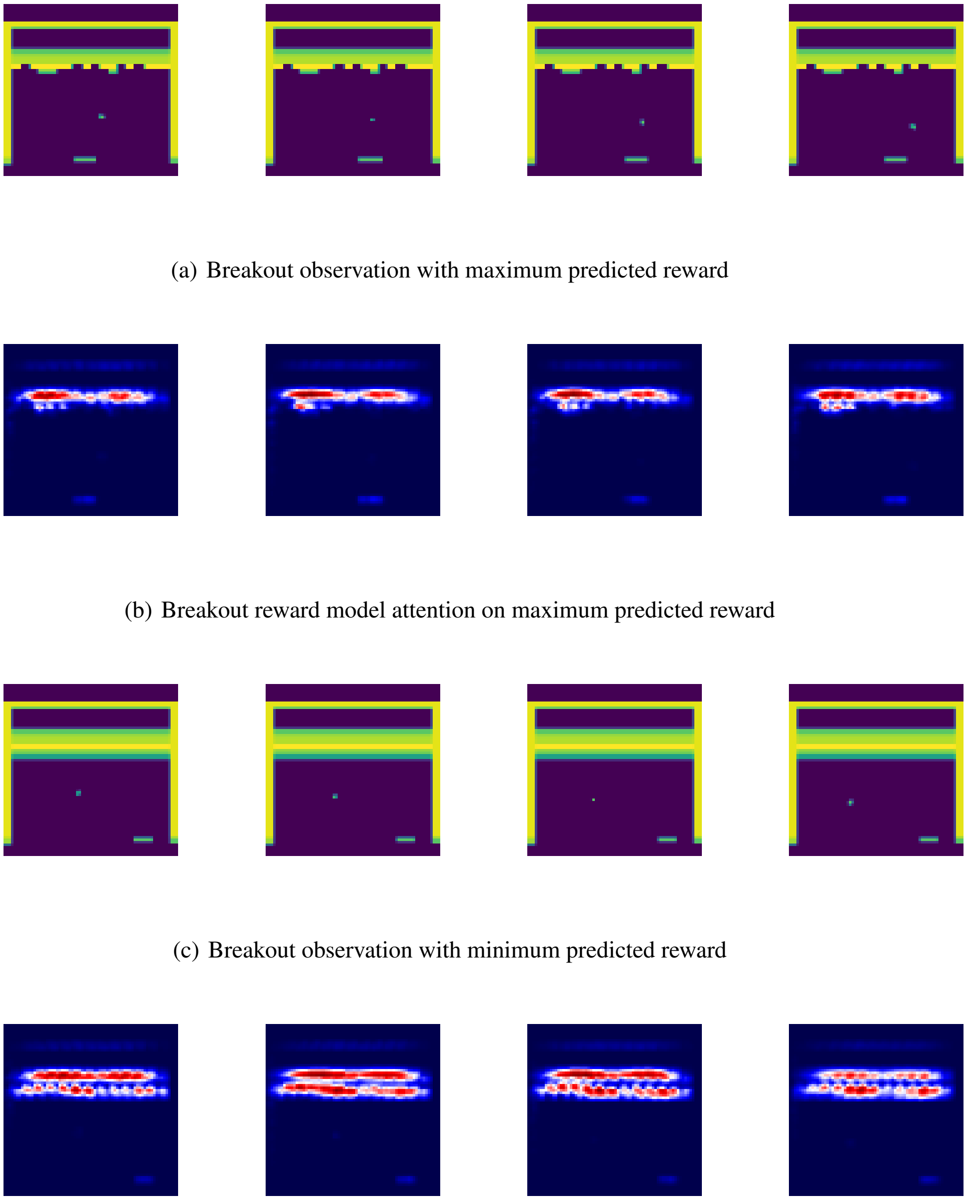
(a) Beam Rider observation with maximum predicted reward

(b) Beam Rider reward model attention on maximum predicted reward

(c) Beam Rider observation with minimum predicted reward

(d) Beam Rider reward model attention on minimum predicted reward

Figure 6: Maximum and minimum predicted observations and corresponding attention maps for Beam Rider across a held-out set of 15 demonstrations. The attention maps show that the reward is a function of the status of the controlled ship as well as the enemy ships and missiles.



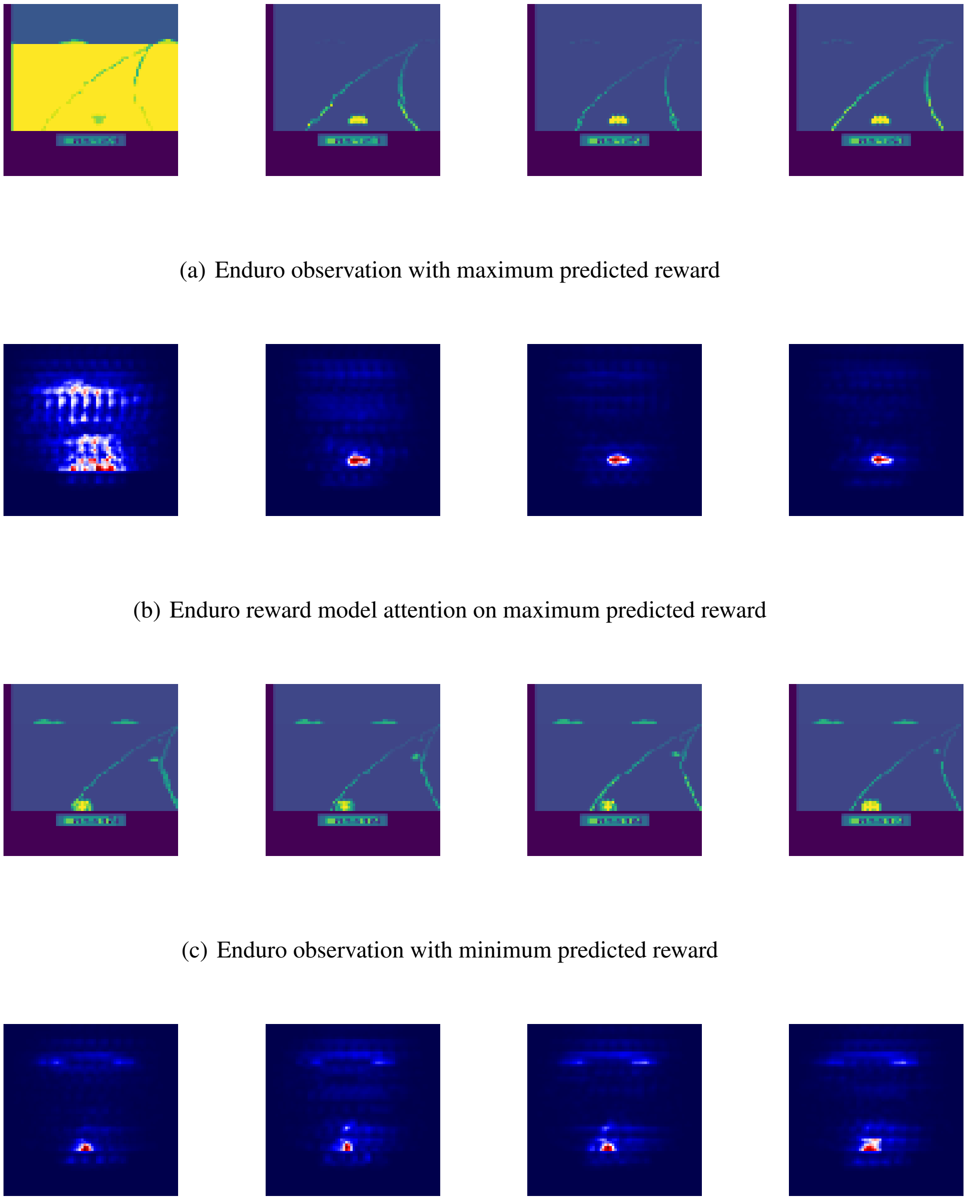
(a) Breakout observation with maximum predicted reward

(b) Breakout reward model attention on maximum predicted reward

(c) Breakout observation with minimum predicted reward

(d) Breakout reward model attention on minimum predicted reward

Figure 7: Maximum and minimum predicted observations and corresponding attention maps for Breakout across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows many of the bricks destroyed. The network has learned to put most of the reward weight on the remaining bricks. The observation with minimum predicted reward is an observation where none of the bricks have been destroyed.



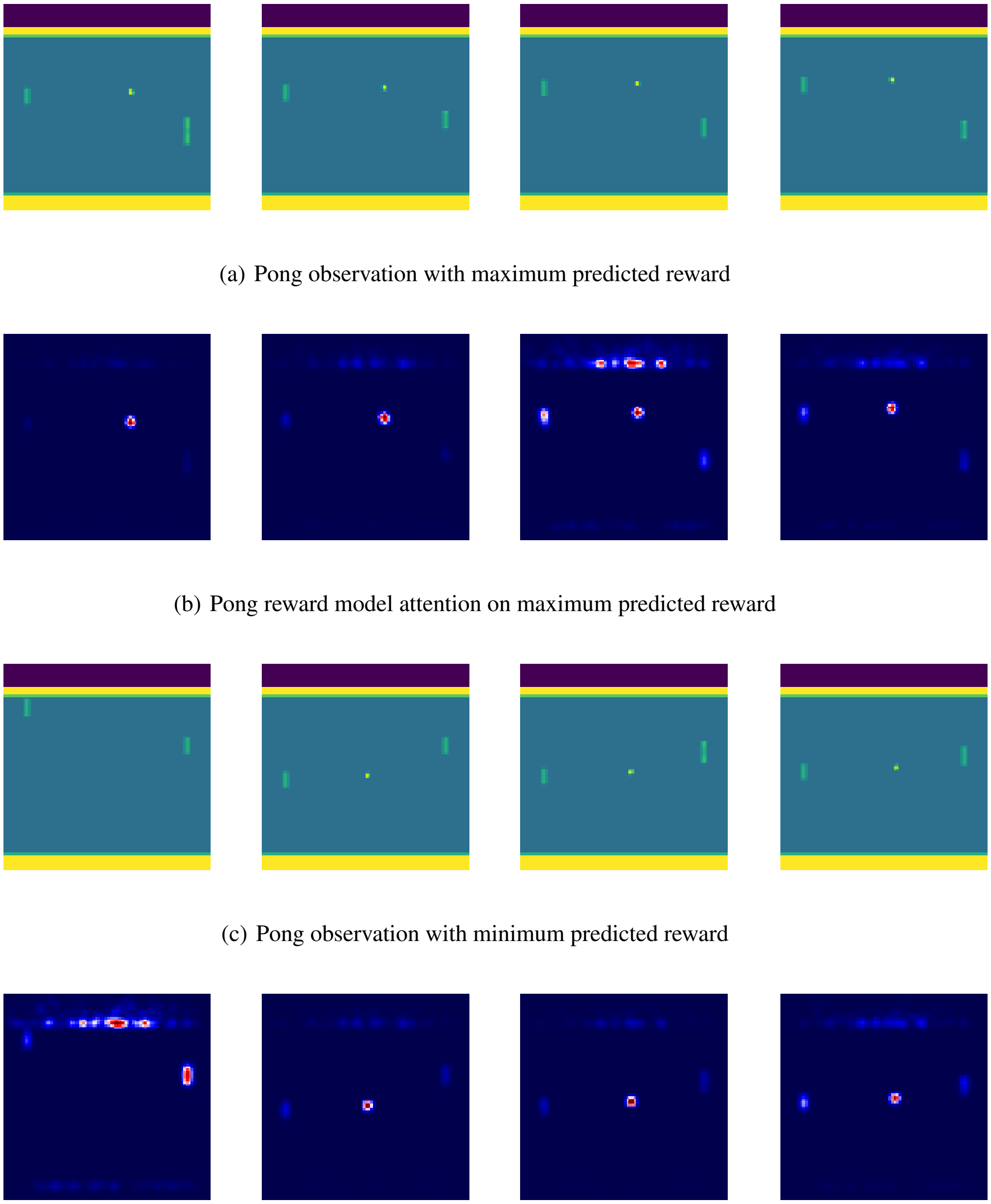
(a) Enduro observation with maximum predicted reward

(b) Enduro reward model attention on maximum predicted reward

(c) Enduro observation with minimum predicted reward

(d) Enduro reward model attention on minimum predicted reward

Figure 8: Maximum and minimum predicted observations and corresponding attention maps for Enduro across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows the car passing from one section of the race track to another as shown by the change in lighting. The observation with minimum predicted reward shows the controlled car falling behind another racer with attention focusing on the car being controlled as well as the speedometer.



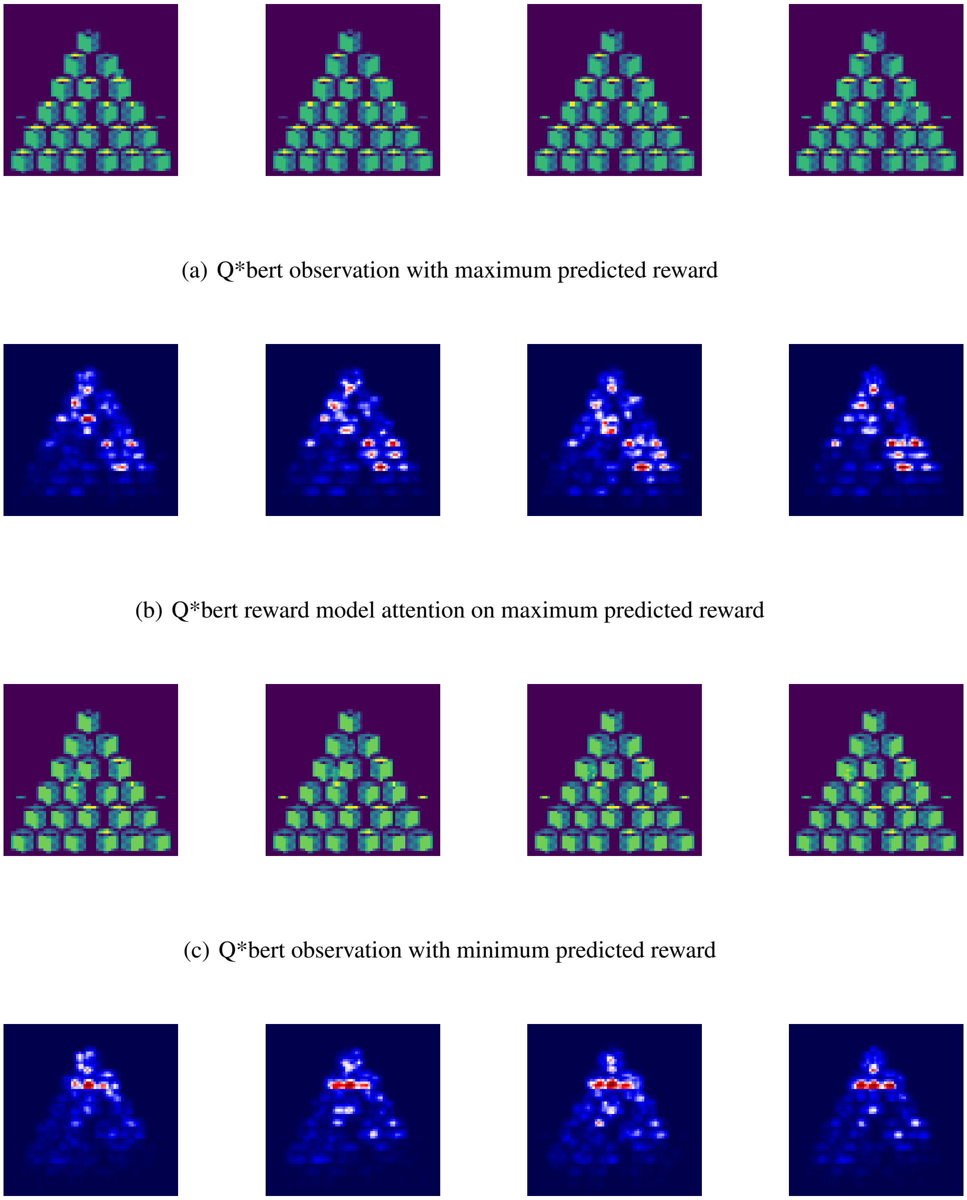
(a) Pong observation with maximum predicted reward

(b) Pong reward model attention on maximum predicted reward

(c) Pong observation with minimum predicted reward

(d) Pong reward model attention on minimum predicted reward

Figure 9: Maximum and minimum predicted observations and corresponding attention maps for Pong across a held-out set of 15 demonstrations. The network attends to the ball and paddles along with some artifacts outside the playing field. The observation with minimum predicted reward shows the ball being sent back into play after the opponent has scored.



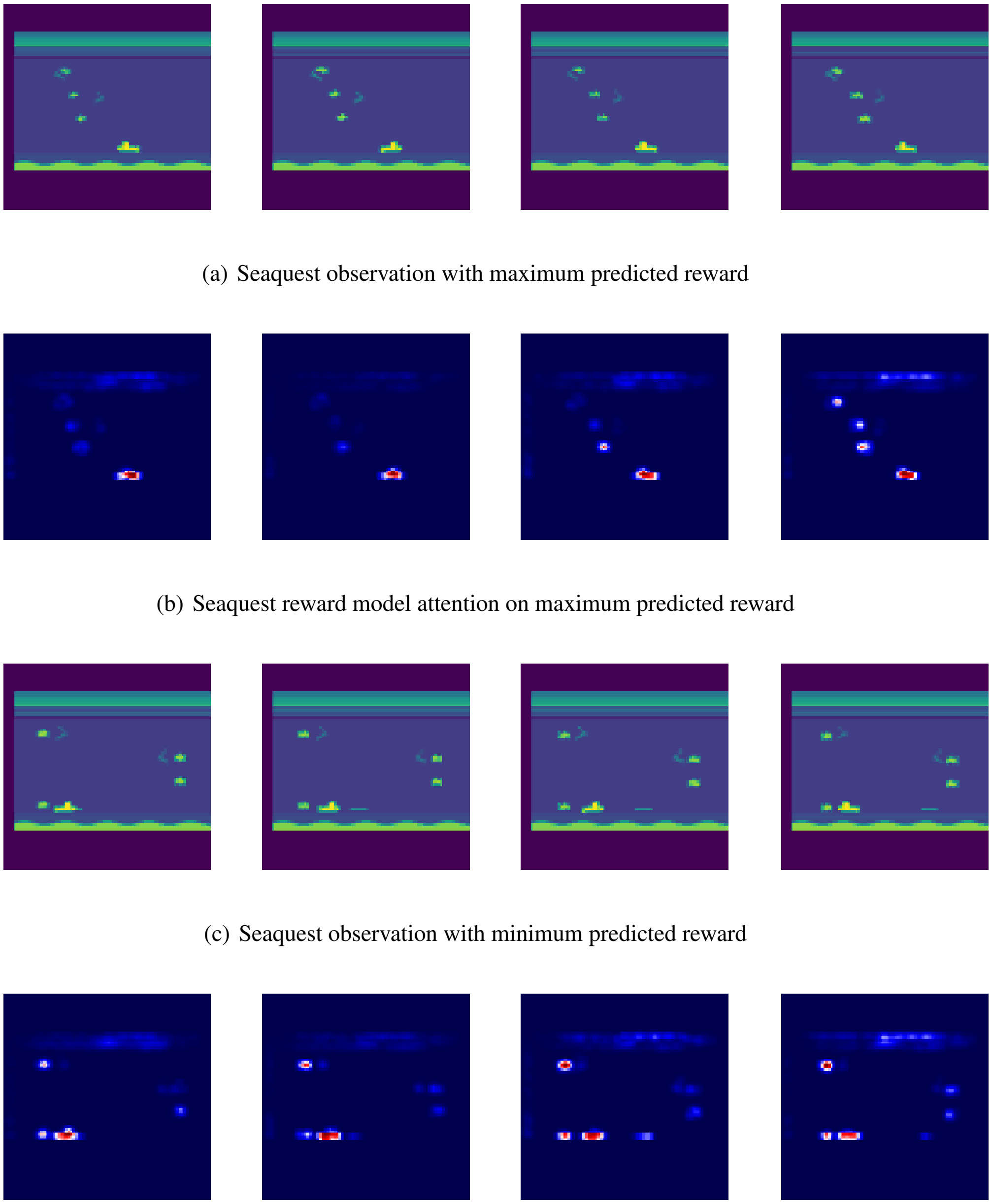
(a) Q*bert observation with maximum predicted reward

(b) Q*bert reward model attention on maximum predicted reward

(c) Q*bert observation with minimum predicted reward

(d) Q*bert reward model attention on minimum predicted reward

Figure 10: Maximum and minimum predicted observations and corresponding attention maps for Q*bert across a held-out set of 15 demonstrations. The network attention is focused on the different stairs, but is difficult to attribute any semantics to the attention maps.



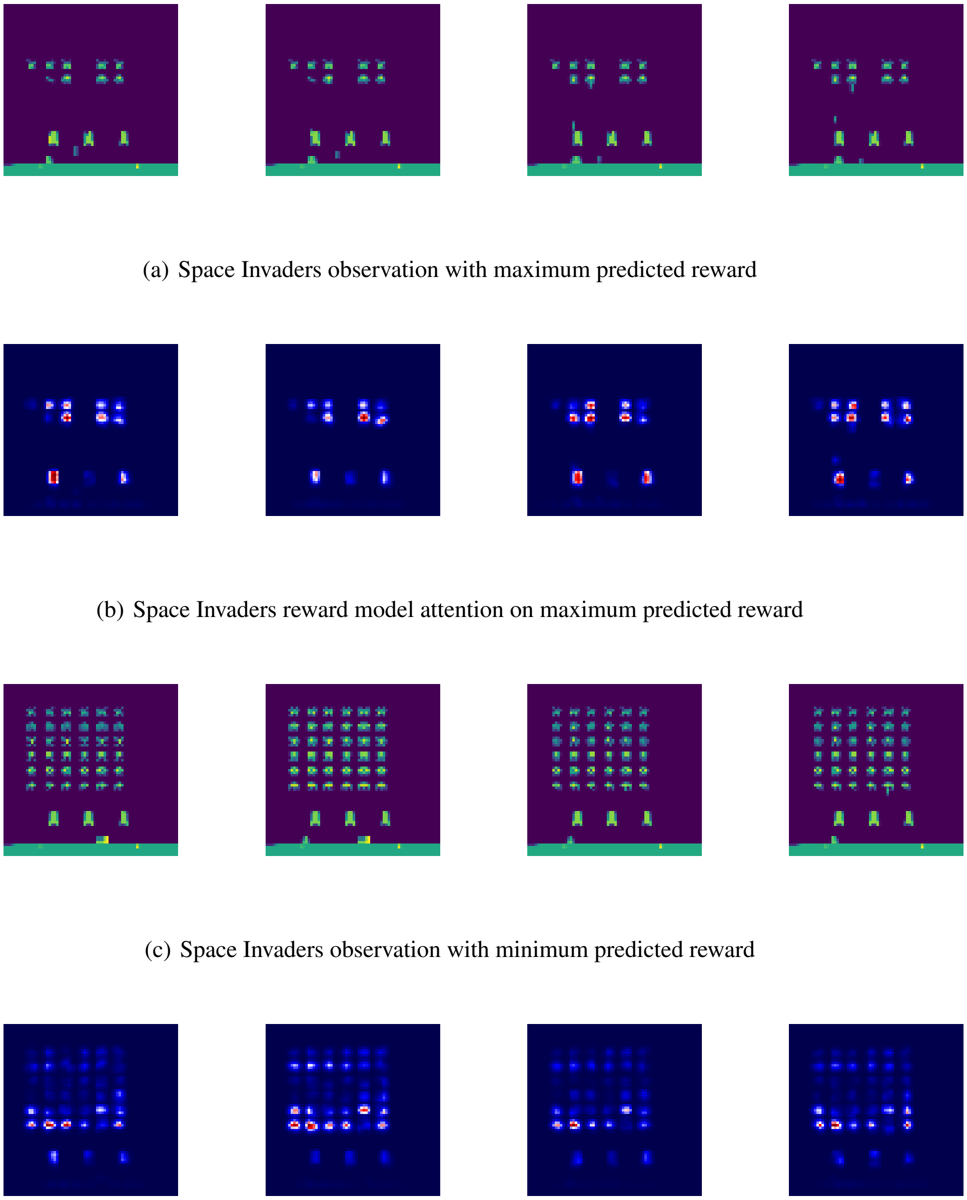
(a) Seaquest observation with maximum predicted reward

(b) Seaquest reward model attention on maximum predicted reward

(c) Seaquest observation with minimum predicted reward

(d) Seaquest reward model attention on minimum predicted reward

Figure 11: Maximum and minimum predicted observations and corresponding attention maps for Seaquest across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows the submarine in a safe location with no immediate threats. The observation with minimum predicted reward shows the submarine one frame before it is hit and destroyed by an enemy shark. This is an example of how the network has learned a shaped reward that helps it play the game better than the demonstrator. The network has learned to give most attention to nearby enemies and to the controlled submarine.



(a) Space Invaders observation with maximum predicted reward

(b) Space Invaders reward model attention on maximum predicted reward

(c) Space Invaders observation with minimum predicted reward

(d) Space Invaders reward model attention on minimum predicted reward

Figure 12: Maximum and minimum predicted observations and corresponding attention maps for Space Invaders across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows an observation where most of the all the aliens have been successfully destroyed and the protective barriers are still intact. The attention map shows that the learned reward function is focused on the barriers and aliens, with less attention to the location of the controlled ship. The observation with minimum predicted reward shows the very start of a game with all aliens still alive. The network attends to the aliens and barriers, with higher weight on the aliens and barrier closest to the space ship.