iD-REX: Iterated Disturbance-based Reward Extrapolation

Jazib Ahmad
Department of Computer Science
University of Toronto
Toronto, Canada
jazibahmad@cs.toronto.edu

Geoffrey H. Harrison
Department of Computer Science
University of Toronto Canada
ghharrison@cs.toronto.edu

Anh Tuan Tran
Department of Computer Science
University of Toronto
Toronto, Canada
alantran@cs.toronto.edu

Abstract: Imitation learning and inverse reinforcement learning methods often suffer from an inability to exceed the performance of the demonstrations they were trained upon. Recent results demonstrate that better-than-demonstration performance can be achieved by extrapolating a reward function from a series of ranked demonstration trajectories and optimizing for the learned reward through reinforcement learning, and further that a series of ranked demonstrations can be generated automatically by injecting noise into an initial policy obtained through behavioral cloning. We extend this algorithm by introducing an iterative ranking-based imitation learning algorithm, Iterated Disturbance-Based Reward Extrapolation (iD-REX), that generates successive sets of demonstrations by performing Behavioral Cloning on the learned policies from the previous iteration. Injecting noise into the new demonstrations results in 3 additional sets of trajectories that are ranked automatically by the amount of noise (but not compared across sets.) In this way, we aim to use iD-REX to iteratively improve an estimated reward function which may be optimized using reinforcement learning. Empirically validating our approach, we achieve a 24% and 201% increase in average performance compared to D-REX in the Mujoco Hopper and Half-Cheetah environments respectively.

1 Introduction

Many practical robotics tasks are too complex to be optimally performed by a human operator, and higher performance may be achieved through use of an autonomous robot. Reinforcement Learning (RL) methods allow a robotic agent to learn how to perform such a task by training a policy that optimizes a reward function that provides feedback for the agent at state-action pairs. However, most real-world applications do not have an innate and easily accessible reward function, and designing one that accurately encapsulates all aspects of the task at hand proves difficult if not impossible.

A common alternative is the imitation learning (IL) paradigm, in which the learner obtains its policy by training to imitate a set of expert demonstrations. A specific form of IL called Inverse Reinforcement Learning (IRL) involves using the expert demonstrations to infer the latent reward function that the expert sought to optimize [1], and then using the learned reward function to train RL methods. However, IRL methods are not without their own pitfalls.

Firstly, owing to the complexity of many robotic tasks, *truly optimal demonstrations are difficult to obtain.* Sub-optimal demonstrations may be used with IRL methods, but doing so introduces another pitfall of IRL: as IRL typically assumes that the expert performed demonstrations that maximize the latent reward, policies obtained via inverse reinforcement learning are typically *unable to outperform their expert.* In the event of expert optimality this is not a serious issue, but in the more realistic case that the demonstrations are sub-optimal, this limits the learned policy to sub-optimal performance as well.

Recent work seeks to overcome these shortcomings. One such method is Trajectory-ranked Reward Extrapolation (T-REX), which uses a ranked set of sub-optimal demonstration trajectories to learn a reward function that, when paired with a reinforcement learning algorithm, gives rise to better-than-demonstrator performance [2]. T-REX does require a human to rank the demonstrations manually, which may be time-consuming or arbitrary, and also requires an adequate number of demonstrations to obtain a meaningful ranking in the first place. A follow-up on the same idea, Disturbance-based Reward Extrapolation (D-REX) [3], improves the algorithm by eliminating the need for manual intervention. D-REX begins with a small number of human demonstrations and obtains an initial policy via behavioral cloning (BC). By injecting increasing amounts of random actions according to a noise schedule, the policy's performance degrades roughly linearly, giving rise to an approximately ranked set of sub-optimal demonstration trajectories.

We propose an extension of this algorithm, Iterated Disturbance-Based Reward Extrapolation (iD-REX). iD-REX seeks to generalize and make iterative the intuition behind D-REX, using newly obtained policy iterations as the source of new ranked sub-optimal demonstrations, improving the reward function estimate and obtaining a new better-than-demonstrator policy with each repetition. We empirically validate the approach on the HalfCheetah-v2 and Hopper-v2 simulated robotics environments in Mujoco, ultimately obtaining learned policies that not only outperform the demonstrations and standard imitation learning, but also outperform the results of D-REX.

2 Related Work

IRL has risen out of the desire to apply RL to tasks in which manually creating the reward function is not feasible or fundamentally impossible, but expert demonstrations of the task are available. The purpose of IRL is to try to learn the reward function that is being optimized by trajectories from an expert demonstrator. This section reviews some of the existing work in IRL.

Maximum Entropy IRL, proposed by Ziebart et al. [4], uses the Principle of Maximum Entropy to learn a reward function that maximizes the likelihood of expert trajectories [4]. Wulfmeier et al. [5] propose using Maximum Entropy based Deep IRL and show that it scales well on a large driving dataset. Finn et al. [6] then build on this idea and make it easier to learn the cost function under unknown dynamics and high dimensionality by introducing Inverse Optimal Control. They show an improvement in both scalability and sample efficiency [6].

However, the challenge with these approaches is that they rely on running RL in an inner loop of IRL, making them very computationally expensive [7]. In addition, sub-optimal demonstrations can cause these algorithms to produce a poor result. Therefore, Sadigh et al. [8] propose Active Preference Based Learning of Reward Functions. This method generates trajectories, asks an expert for preferences on those trajectories and determines a reward function by assigning a higher reward to the trajectories that the expert considers better [8]. It eliminates the inner loop, and is not constrained by the performance of sub-optimal demonstrators. However, it does create the need to query the expert on each iteration of the algorithm.

Ho and Ermon [7] propose Generative Adversarial Imitation Learning (GAIL). GAIL eliminates the need to run RL in an inner loop, without creating the need to query the expert for preferences by using Generative Adversarial Networks (GAN) [7]. The goal of the generator is to generate trajectories that match the expert trajectories, as judged by the discriminator [7]. Because of the existance of the discriminator, the generator is able to determine which trajectories to prefer without human supervision [7]. However, GAIL by its very nature only attempts to mimic the expert trajectories and is therefore still constrained by sub-optimal demonstrators [2] [3].

To address this shortcoming, Brown et al. [2] propose T-REX, an algorithm that uses ranked demonstrations to infer a reward function that is then used to learn a policy that performs better than the demonstrations [2]. Unlike Active Preference Based Learning of Reward Functions [8], T-REX does not generate new trajectories. Rather it uses preferences on existing trajectories, allowing the algorithm to run without active involvement from the expert and potentially improving sample efficiency. As a follow up to this work, Brown et al. [3] then propose D-REX, which works similarly to T-REX, but can be applied to cases where trajectory rankings are unavailable [3]. D-REX trains an initial policy using Behavioural Cloning and uses it with different levels of noise to create the ranked trajectories [3]. Brown et al. [3] show theoretically and empirically that this approach can result in better performance than the demonstrator.

3 Methodology

As iD-REX is an extension of D-REX, many of the same theoretical conditions apply. Addressing a terminological prerequisite of D-REX, consider a learner that approximates the true reward function of the environment, R^*, with a linear combination of features $R(s) = w^T\phi(s)$ for a state s. The expected return of a policy evaluated on $R(s)$ is therefore given as

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

Φ_π is the expected discounted feature counts that arise when obeying the policy π. Brown et al. then describe a Theorem [3] that justifies the intuition of D-REX:

Theorem 1: *If the estimated reward function is $\hat{R}(s) = w^T\phi(s)$, the true reward function is $R^*(s) = \hat{R}(s) + \epsilon$ for some error function $\epsilon : \mathcal{S} \to R$, 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 $\|\epsilon\|_\infty = sup\{|\epsilon(e)| : s \in \mathcal{S}$
Put simply, extrapolation beyond the demonstrator is guaranteed when the demonstrator is truly suboptimal, the error of the learned reward function compared to the true reward function is small, and the set of states explored by the learned policy $\hat{\pi}$ are close to that explored by the optimal policy. Brown et al. also note [3] that if the learned reward function is close to the true reward function, ϵ_Φ can be made small via reinforcement learning, and therefore the success of the algorithm depends primarily on improving the accuracy of the learned reward function via ranked demonstrations.

Brown et al. additionally prove a proposition [3] that using a preference ranking-based method over the demo trajectories leads to more accurate estimates of the true reward function R^*:

Proposition 1: *There exist Markov Decision processes (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*

$$\pi_E = \underset{\pi\in\Pi}{\arg\max}\, J(\pi|R^*) \text{ and } J(\pi_1|R^*) \ll J(\pi_2|R^*) \tag{3}$$

$$\pi_E = \underset{\pi\in\Pi}{\arg\max}\, J(\pi|\hat{R}) \text{ and } J(\pi_1|\hat{R}) = J(\pi_2|\hat{R}) \tag{4}$$

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

$$\pi_E = \underset{\pi\in\Pi}{\arg\max}\, J(\pi|\hat{R}) \text{ and } J(\pi_1|\hat{R}) < J(\pi_2|\hat{R}) \tag{5}$$

The first two above equations describe a Markov Decision Process where optimizing a policy for the approximated reward function results in an optimal policy, yet the approximated reward function may be wildly inaccurate on other policies. This, particularly in cases where the expert is

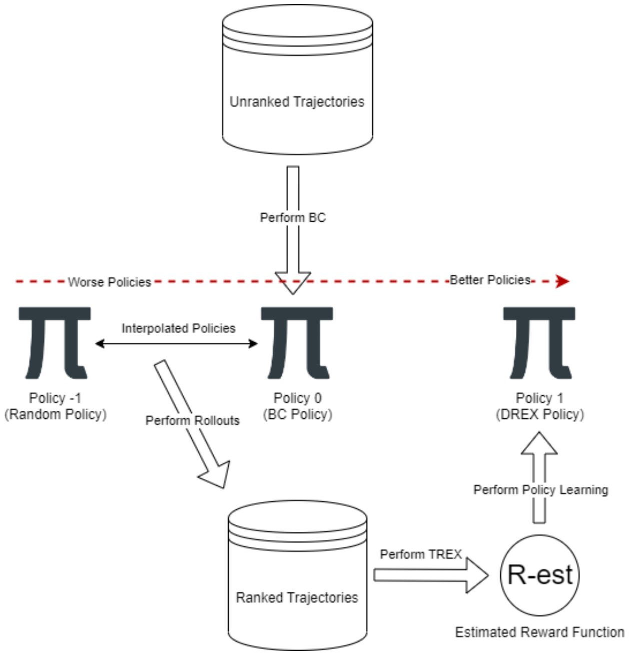


Figure 1: D-REX

sub-optimal, can result in learned reward functions that overfit to ”expert” mistakes and generalize poorly. However, the issue is mitigated by the enforcement of a preference ranking over the trajectories.

3.1 iD-REX: The probabilistic interpolation approach

The intuition behind the generation of the automatically-ranked sub-optimal demonstration set in D-REX is that adding random actions to a behavioral cloning policy π_{BC} according to a noise schedule $\mathcal{E}$ ”interpolates” between the behavioral cloning policy and a random policy π_{random}. By generating a set of such ”interpolated” policies and performing rollouts on each one according to a hyperparameter $\mathcal{N}$, a set of trajectories can be obtained that is automatically ranked by the amount of noise that was injected into the behavioral cloning policy that generated the trajectories.

We aim to make this process iterative by producing new automatically ranked sets of sub-optimal demonstrations to augment that created by D-REX. To achieve this, we must generalize the notion of ”interpolation” between policies introduced in D-REX. We define a form of *probabilistic policy interpolation* that interpolates between two iterations of learned policy by selecting actions from the two policies according to a parameter α:

$$\forall \alpha \in \mathcal{A} \qquad \pi_k(\cdot|\alpha) = \left\{ \begin{array}{lr} \pi_k(\cdot) & \text{probability } \alpha \\ \pi_{k-1}(\cdot) & \text{probability } 1-\alpha \end{array} \right\} \tag{6}$$

(For notational convenience, we denote the policy learned by D-REX as π_1, the behavioral cloning policy obtained from the initial demonstrations as π_0, and the random policy as π_{-1}.) On a given iteration of D-REX k, using a set of action selection probabilities $\mathcal{A} = (\alpha_1, \alpha_2, ...\alpha_d)$, we generate a set of interpolated policies $\{\pi_k(\cdot|\alpha_i)\}$ and perform $\mathcal{N}$ rollouts on each one, obtaining $\mathcal{N} \times d$ new sub-optimal demonstrations. These may be ranked according to:

$$\tau_i \succ \tau_j : \tau_i \sim \pi_k(\cdot|\alpha_i), \tau_j \sim \pi_k(\cdot|\alpha_j), \alpha_i > \alpha_j \tag{7}$$

That is, for two trajectories generated via probabilistic policy interpolation between the same two endpoint policies, the trajectory generated with a higher α (equivalently, the trajectory generated

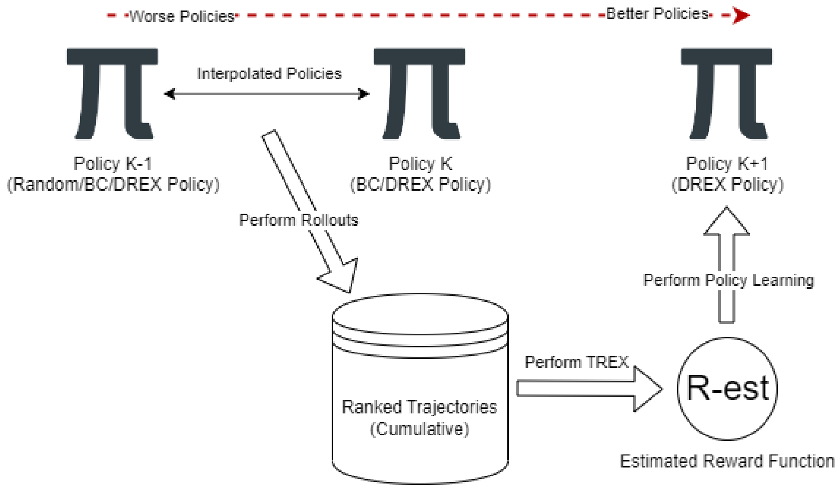


Figure 2: iD-REX: The Probabilistic Interpolation Approach

by a policy taking more actions from a newer policy iteration) is ranked as superior. Additionally, trajectories generated in later iterations of iD-REX are ranked as superior to those from earlier iterations, as intuitively later policy iterations should produce better performance and therefore better noise-injected demonstrations than earlier iterations:

$$\tau_i \succ \tau_j : \tau_i \sim \pi_k(\cdot|\alpha_i), \tau_j \sim \pi_{k-dk}(\cdot|\alpha_j), dk > 0 \tag{8}$$

We have obtained a method to augment the automatically ranked sub-optimal demonstration set with new automatically ranked demonstrations, and to rank the new demonstrations when compared to the old. Enforcing this preference should, according to Proposition 1, allow learning of a reward function that may be optimized for to obtain a more optimal policy, as suggested by Theorem 1. This gives rise to the full iD-REX algorithm:

Algorithm 1 iD-REX with probabilistic policy interpolation

```
Require: Demonstrations $\mathcal{D}$, action selection probabilities $\mathcal{A}$, number of rollouts $\mathcal{N}$, number of iterations $\mathcal{K}$
Run D-REX on $\mathcal{D}$ to obtain a learned policy $\pi_1$ and preference-ranked trajectory set $\mathcal{T}$
for k $\in$ (1, 2, ..., $\mathcal{K} - 1$) do
    for all $\alpha \in \mathcal{A}$ do
        Define $\pi_k(\cdot|\alpha)$ according to Equation 6
        Generate a set of $\mathcal{N}$ trajectories according to $\pi_k(\cdot|\alpha)$
    end for
    Generate automatic preference labels $\tau_i \succ \tau_j : \tau_i \sim \pi_k(\cdot|\alpha_i), \tau_j \sim \pi_k(\cdot|\alpha_j), \alpha_i > \alpha_j$ or $\tau_i \sim \pi_k(\cdot|\alpha_i), \tau_j \sim \pi_{k-dk}(\cdot|\alpha_j), dk > 0$ and augment $\mathcal{T}$
    Run T-REX on $\mathcal{T}$ to obtain reward function estimate $\hat{R}$
    Obtain policy $\pi_{k+1}$ through reinforcement learning with reward $\hat{R}$
end for
Return $\pi_{\mathcal{K}}$
```

Unfortunately, this form of iD-REX did not produce the desired results (see Evaluation for details). In general, the probabilistically interpolated policy $\pi_k(\cdot|\alpha)$ did not produce linearly scaling performance in between that of policies π_k and π_{k-1} as expected; frequently, the interpolated policy generated poorer returns than either ”endpoint” policy actions were taken from. This resulted in automatic trajectory rankings that were incorrect, resulting in a worse estimation of the reward function and therefore violating an assumption of Theorem 1.

3.2 iD-REX: The action noise approach

We seek to address a shortcoming in the previous approach in that the automatically generated rankings were frequently incorrect, resulting in an inaccurate estimate of the reward function being learned. A reason for this appears to have been that policies learned via Proximal Policy Optimization, our choice of reinforcement learning algorithm, are highly sensitive to disturbances, particularly when actions from another policy are randomly injected.

To address this, we considered an approach in which random noise is added to the actions generated by the policy, rather than randomly selecting actions from a different policy. Using Gaussian random noise, this is done as follows:

$$\forall \epsilon \in \mathcal{E} \qquad \pi_k(\cdot|\epsilon) = \pi_k(\cdot) + \epsilon \cdot \mathcal{N}(0, \sigma^2) \tag{9}$$

where σ^2 is the ones vector in the dimension of the action space.

Doing away with the notion of ”interpolating” between policies, this approach sought to improve the estimate of the reward function by exposing the reward function learner to a greater set of ranked trajectories covering more of the policy space, particularly for higher-scoring trajectories that could not be obtained through behavioral cloning. This gives rise to the second version of iD-REX, described in **Algorithm 2**. Algorithm 2 may also be defined using another form of action noise, such as auto-correlated noise following an Ornstein-Uhlenbeck process or time-correlated noise.

Algorithm 2 iD-REX with Gaussian action noise

```
Require: Demonstrations $\mathcal{D}$, action noise schedule $\mathcal{E}$, number of rollouts $\mathcal{N}$, number of iterations $\mathcal{K}$
Run D-REX on $\mathcal{D}$ to obtain a learned policy $\pi_1$ and preference-ranked trajectory set $\mathcal{T}$
for k $\in$ (1, 2, ..., $\mathcal{K} - 1$) do
    for all $\epsilon \in \mathcal{E}$ do
        Define $\pi_k(\cdot|\epsilon)$ according to Equation 9
        Generate a set of $\mathcal{N}$ trajectories according to $\pi_k(\cdot|\epsilon)$
    end for
    Generate automatic preference labels $\tau_i \succ \tau_j : \tau_i \sim \pi_k(\cdot|\epsilon_i), \tau_j \sim \pi_k(\cdot|\epsilon_j), \epsilon_i < \epsilon_j$ and augment $\mathcal{T}$
    Run T-REX on $\mathcal{T}$ to obtain reward function estimate $\hat{R}$
    Obtain policy $\pi_{k+1}$ through reinforcement learning with reward $\hat{R}$
end for
Return $\pi_{\mathcal{K}}$
```

This approach to iD-REX also did not produce policies that outperform the demonstrator (again, see Evaluation for detailed results and explanation). While the automatic rankings were this time correct, as adding noise to the actions themselves instead of injecting actions into a policy from a different policy degraded the returns of the generated trajectories roughly linearly, policies learned by this approach did not produce higher returns than the demonstrators. The reinforcement learning step converged very quickly and the resulting trajectories had a low standard deviation, suggesting that very little of the policy space was explored and a local optimizer was reached. Use of alternative forms of action noise, such as Ornstein-Uhlenbeck noise or time-correlated noise, did not remedy these failures.

3.3 iD-REX: The multi-behavioral cloning approach

Following the unsatisfactory results of the two prior approaches to iD-REX, a third attempt was made following a pattern more closely related to that of D-REX. Using trajectories generated by Proximal Policy Optimization policies as the source of the ranked sub-optimal demonstration set, whether with or without noise distubing the policy actions, consistently appeared to produce one of two pitfalls: either the policy was *highly sensitive to perturbations* and would not degrade in a linear

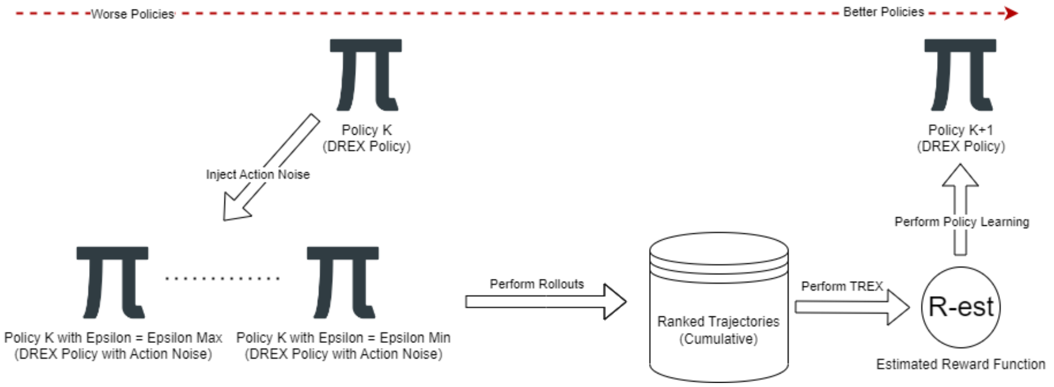


Figure 3: iD-REX: The Action Noise Approach

fashion as did behavioral cloning policies in D-REX, or that *reward functions trained using these trajectories discourage exploration* and quickly converge to a policy that has poor returns.

To regain the desirable properties exhibited by the noise-injected behavioral cloning policy used as the source of sub-optimal demonstrations in D-REX, we take advantage of the fact that a D-REX run produces multiple policies $\pi^{(i)}$, depending on the hyperparameter $\mathcal{R}$ for the number of reinforcement learning runs. For each such policy, we generate one demonstration trajectory and perform BC upon it to gain a new policy.

For each behavioral cloning policy π_{BC}, noise in the form of random actions can be injected as in D-REX:

$$\forall \epsilon \in \mathcal{E} \qquad \pi_{BC}(\cdot|\epsilon) = \left\{ \begin{array}{ll} \pi_{BC}(\cdot) & \text{probability } 1-\epsilon \\ \pi_{random}(\cdot) & \text{probability } \epsilon \end{array} \right\} \tag{10}$$

We obtain multiple behavioral cloning policies, one cloned from each policy obtained via the previous interation of reinforcement learning runs, and inject random actions into them according to a noise schedule $\mathcal{E} = (\epsilon_1, \epsilon_2, ..., \epsilon_d)$ as in D-REX. For every $\epsilon_i \in \mathcal{E}$ we perform N rollouts on the noise-injected policies $\pi_{BC}^{(i)}$, ultimately obtaining $N \times d \times \mathcal{R}$ new sub-optimal demonstrations. These demonstrations can be ranked in a similar fashion to those used in D-REX:

$$\tau_i \succ \tau_j : \tau_i \sim \pi_{BC}^{(i)}(\cdot|\epsilon_i), \tau_j \sim \pi_{BC}^{(i)}(\cdot|\epsilon_j), \epsilon_i < \epsilon_j \tag{11}$$

That is, one trajectory is ranked as superior to another if the proportion of random actions in its generating policy is lower. Note that these rankings are *only* applied to pairs of trajectories that originated from the same behavioral cloning policy $\pi_{BC}^{(i)}$; as the undisturbed behavioral cloning policies do not all have the same performance, the general pattern that a trajectory generated with higher amounts of noise will have worse performance than one with a lower amount of noise may not hold when comparing trajectories originating from different behavioral cloning policies.

We then obtain an algorithm for iD-REX that uses multiple behavioral cloning policies cloned from the previous iteration of learned policy to obtain a new set of automatically ranked sub-optimal demonstrations, displayed in **Algorithm 3**.

4 Evaluation

4.1 iD-REX: The probabilistic interpolation approach

To evaluate that our algorithm works successfully, we must validate firstly that our method of generating sub-optimal demonstrations results in an improved estimate of the reward function, and then

Algorithm 3 iD-REX with multiple behavior cloning policies

```
Require: Demonstrations $\mathcal{D}$, noise schedule $\mathcal{E}$, number of rollouts $\mathcal{N}$, number of iterations $\mathcal{K}$, number of reinforcement learning runs $\mathcal{R}$
Run D-REX on $\mathcal{D}$ to obtain learned policies $\pi_1^{(1)}$, $\pi_1^{(2)}$, ..., $\pi_1^{(\mathcal{R})}$ and preference-ranked trajectory set $\mathcal{T}$
for k $\in$ (1, 2, ..., $\mathcal{K} - 1$) do
    for r $\in$ (1, 2, ..., $\mathcal{R}$) do
        Generate trajectory $\tau$ from policy $\pi_k^{(r)}$
        Perform behavioral cloning on $\tau$ to obtain policy $\pi_{BC}^{(k,r)}$
        for all $\epsilon \in \mathcal{E}$ do
            Generate a set of $\mathcal{N}$ trajectories according to $\pi_{BC}^{k,r}(\cdot|\epsilon)$
        end for
    end for
    Generate automatic preference labels $\tau_i \succ \tau_j : \tau_i \sim \pi_{BC}^{(k,r)}(\cdot|\epsilon_i), \tau_j \sim \pi_{BC}^{(k,r)}(\cdot|\epsilon_j), \epsilon_i < \epsilon_j$ and augment $\mathcal{T}$
    Run T-REX on $\mathcal{T}$ to obtain reward function estimate $\hat{R}$
    Obtain policies $\pi_{k+1}^{(1)}$, $\pi_{k+1}^{(2)}$..., $\pi_{k+1}^{(\mathcal{R})}$ through reinforcement learning with reward $\hat{R}$
end for
Return $\pi_{\mathcal{K}}^{(1)}$, ..., $\pi_{\mathcal{K}}^{(\mathcal{R})}$
```

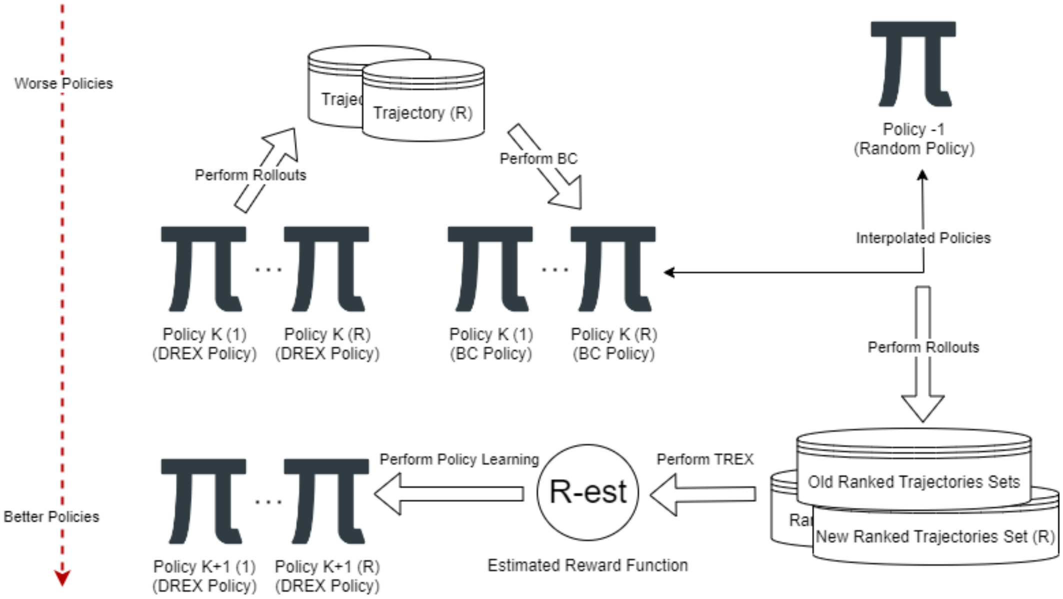
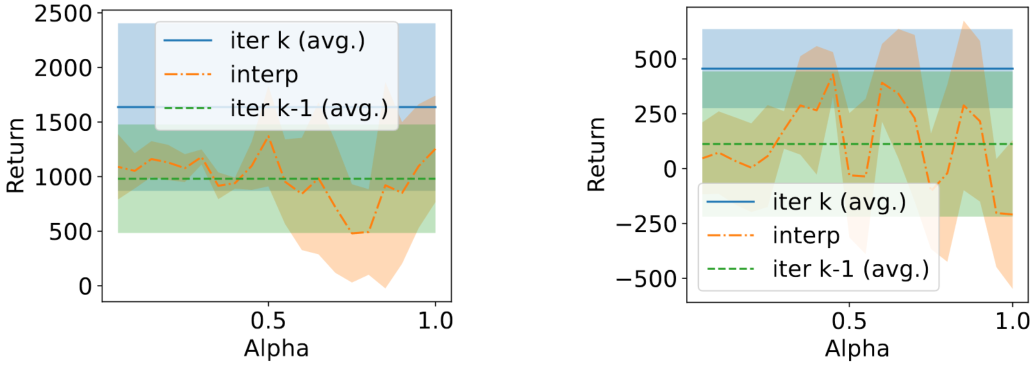


Figure 4: iD-REX: The Multi-BC Approach

that training using reinforcement learning with that reward estimate results in performance better than both the demonstrations and the results of D-REX. In Figure 5, we illustrate the performance degradation charts that arise from using probabilistic interpolation (as described in Methodology) to degrade the performance of the learned policy from D-REX. Contrary to expectations, we do not obtain performance that linearly degrades from that of the learned D-REX policy towards that of the BC policy as α is reduced from 1 to 0. Rather, we end up with highly variable trajectory returns that do not follow any discernible pattern, and indeed are often worse than the average performance of either policy actions were sampled from.

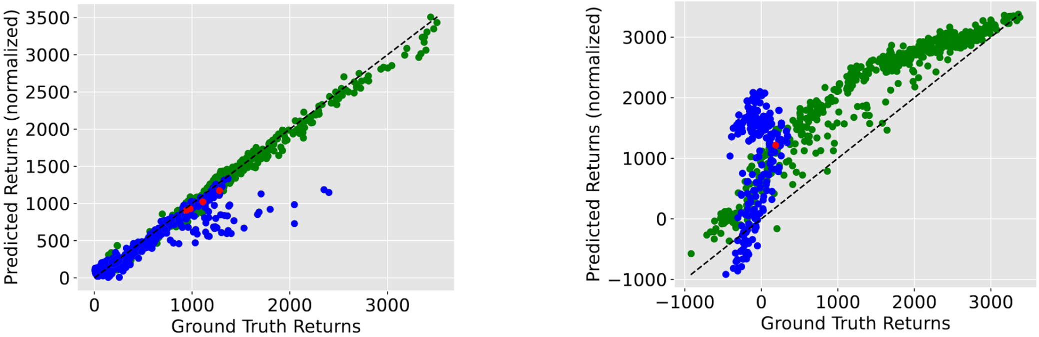


(a) Hopper

(b) HalfCheetah

Figure 5: Degradation plots of returns of the interpolated policies in Hopper and HalfCheetah. The average and standard deviation of performance of the learned D-REX policy is plotted in blue, while the analogous metrics for the behavioral cloning policy are plotted in green.

As the generated trajectories are ranked on the basis of the amount of injected noise, the lack of the expected performance degradation means that the automated rankings become incorrect. This in turn harms the quality of the learned reward function (Figure 6), resulting in a learned policy that is significantly worse than the D-REX policy, as displayed in Table 1. Bold denotes a best average reward across 3 reinforcement learning runs.



(a) Hopper

(b) HalfCheetah

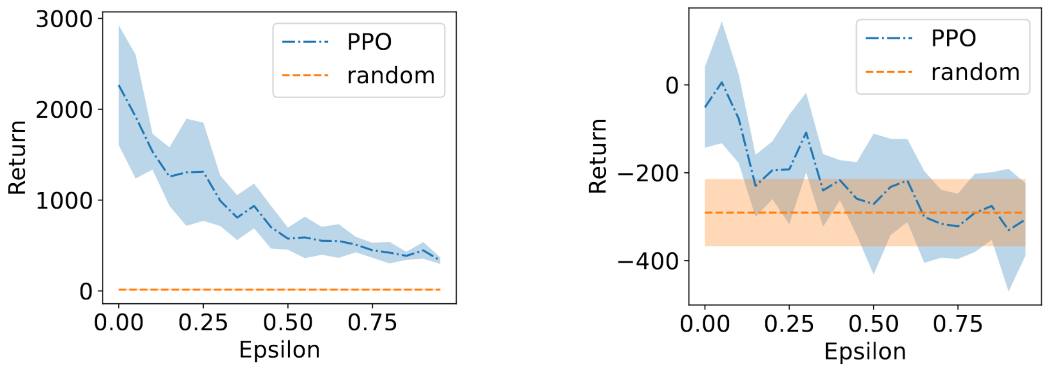
Figure 6: Ground truth vs predicted reward using the probabilistic interpolation approach. Red denotes expert demonstrations, blue denotes the generated sub-optimal demonstration set, and green denotes unseen trajectories not exposed to the learner during reward training.

	Demonstrations		BC		D-REX		iD-REX (Ours)	
Task	Average	Best	Average	Stdev.	Average	Stdev.	Average	Stdev.
Hopper	1029.1	1167.9	1015.2	98.3	**2047.2**	1115.6	339.6	34.4
HalfCheetah	187.7	187.7	-95.6	178.5	**293.5**	526.8	234.50	389.79

Table 1: Comparison of the performance of the probabilistic interpolation variant of iD-REX with the demonstrations, BC, and D-REX. Listed result is for the best average ground-truth trajectory returns over 3 reinforcement learning runs with 20 rollouts per run. Bold denotes returns higher than that of the best demonstration.

4.2 iD-REX: The action noise approach

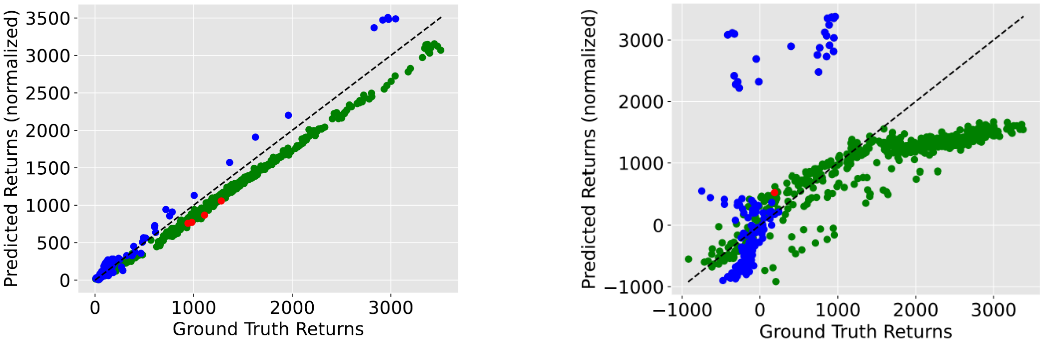
The second attempt at implementing iD-REX did not attempt to interpolate between policies. This approach involved adding Gaussian noise to the actions generated by the learned policy from D-REX, instead of randomly selecting actions from the previous iteration of learned policy, in order to avoid the sharp drops in performance occurring when random actions were injected into the learned policy. While this produced policies whose performance degraded roughly monotonically as desired, and the resulting plots of predicted and true reward for the demo and unseen trajectories appeared to show a strong correlation (Figure 8), training a policy using the learned reward function often obtained results that were no better than the demonstrations.



(a) Hopper

(b) HalfCheetah

Figure 7: Degradation plots of the action noise-based approach for Hopper and HalfCheetah. Compared to the probabilistic interpolation approach, here a more roughly monotonic degradation is achieved.



(a) Hopper

(b) HalfCheetah

Figure 8: Ground truth vs predicted reward using the action noise-based approach. Red denotes expert demonstrations, blue denotes the generated sub-optimal demonstration set, and green denotes unseen trajectories not exposed to the learner during reward training.

The reinforcement learning algorithm, when used with a reward function trained on a ranked dataset of noisy-action trajectories, was observed to converge very quickly and had a low standard deviation, suggesting that little exploration of the policy space was occurring during training. A possible explanation is that, as all of the trajectories in the dataset (except for those generated by D-REX) were generated by Gaussian noise centered on the same policy, the resulting trajectories were all quite similar to each other, and subsequently the learned reward function was trained on very little of the trajectory space and learned to discourage exploration of the policy space. This would cause the reinforcement learning to quickly converge to a local optimizer, albeit one with evidently poor returns.

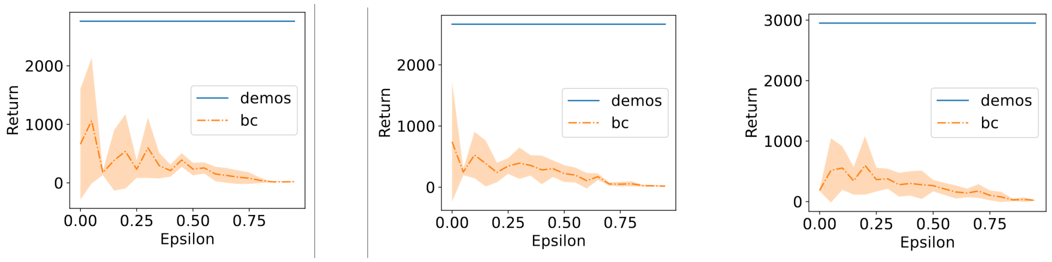
This approach was also repeated with different types of action noise, such as autocorrelated noise following an Ornstein-Uhlenbeck process and (both positively and negatively) time-correlated noise, but it did not improve the performance of the learned policy.

	Demonstrations		BC		D-REX		iD-REX (Ours)	
Task	Average	Best	Average	Stdev.	Average	Stdev.	Average	Stdev.
Hopper	1029.1	1167.9	1015.2	98.3	**2047.2**	1115.6	336.7	7.1
HalfCheetah	187.7	187.7	-95.6	178.5	**293.5**	526.8	-212.0	68.1

Table 2: Comparison of the performance of the action noise-based variant of iD-REX with the demonstrations, BC, and D-REX. Listed result is for the best average ground-truth trajectory returns over 3 reinforcement learning runs with 20 rollouts per run. Bold denotes returns higher than that of the best demonstration.

4.3 iD-REX: The multi-behavioral cloning approach

As described earlier, a revised approach to iD-REX took advantage of the existence of multiple learned policies as a result of the reinforcement learning to iteratively improve upon the results of D-REX. As seen in Figure 9 and Figure 10, the BC policies cloned from each of the reinforcement learning policies experiences a roughly linear degradation in performance with the addition of random actions. Further, as seen in Figure 11, the learned reward function quite closely correlated predicted and ground-truth reward.

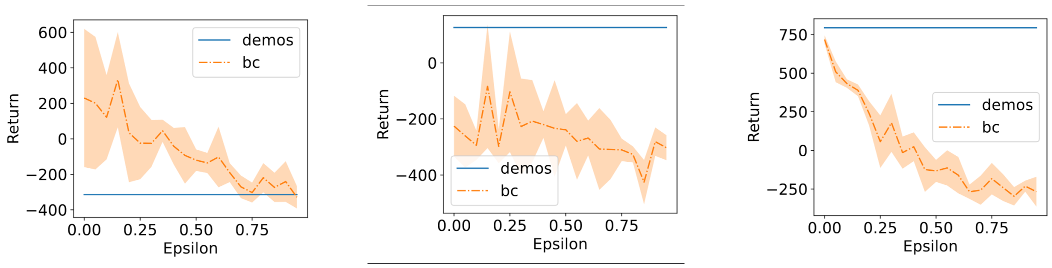


(a) Hopper, run 1

(b) Hopper, run 2

(c) Hopper, run 3

Figure 9: Degradation plots of mean and standard deviation of the three new BC policies in Hopper. The blue line denotes the ground-truth return of the trajectory that the BC policy was trained on.

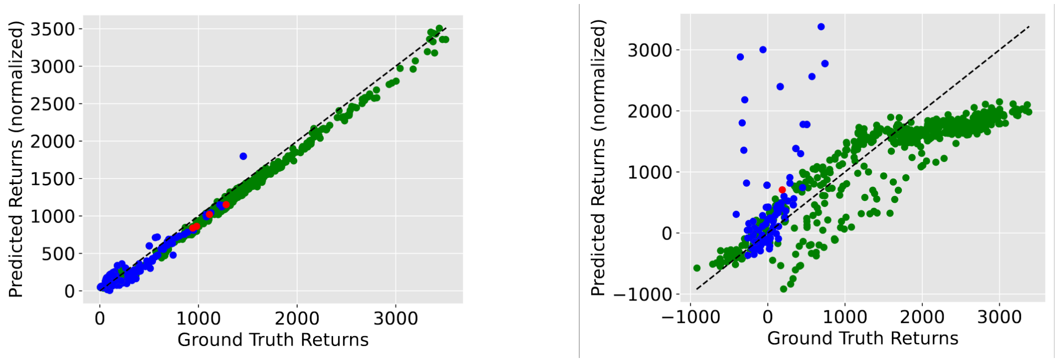


(a) HalfCheetah, run 1

(b) HalfCheetah, run 2

(c) HalfCheetah, run 3

Figure 10: Degradation plots of returns of the three new BC policies in HalfCheetah.



(a) Hopper

(b) HalfCheetah

Figure 11: Ground truth vs predicted reward using the multi-BC approach. Red denotes expert demonstrations, blue denotes the generated sub-optimal demonstration set, and green denotes unseen trajectories not exposed to the learner during reward training.

	Demonstrations		BC		D-REX		iD-REX (Ours)	
Task	Average	Best	Average	Stdev.	Average	Stdev.	Average	Stdev.
Hopper	1029.1	1167.9	1015.2	98.3	**2047.2**	1115.6	**2527.8**	745.0
HalfCheetah	187.7	187.7	-95.6	178.5	**293.5**	526.8	**885.0**	93.1

Table 3: Comparison of the performance of the multi-BC variant of iD-REX with the demonstrations, BC, and D-REX. Listed result is for the best average ground-truth trajectory returns over 3 reinforcement learning runs with 20 rollouts per run. Bold denotes returns higher than that of the best demonstration.

Performing a reinforcement learning run with the new reward function estimate obtained much better results than had been achieved with earlier forms of iD-REX. We observe in Table 3 that using the multi-behavioral cloning approach to iD-REX, we obtain a best average reward across the three reinforcement learning runs that is better than not only the best demonstration, but also the performance of D-REX, providing an improvement of 24% when evaluated on the Hopper environment and a 201% improvement on the HalfCheetah environment. Further, the standard deviation of the iD-REX rollouts is far lower than that of D-REX, suggesting that iD-REX is also more consistent in its performance than D-REX.

5 Limitations and Future Work

This project encountered a number of limitations and opportunities for further investigation. Firstly, iD-REX was only evaluated on two Mujoco environments due to time and resource constraints. Therefore, while these results provide some evidence for the effectiveness of our approach, more evaluation on other environments are needed to conclude that our approach can outperform D-REX on a diverse set of environments. Specifically, more evaluation on environments with a more complicated ground truth reward function may show a more significant improvement in performance, or that the method is not effective on complicated environments.

The multi-behavioural cloning variant of iD-REX requires trajectories that perform worse than the previous iteration of the policy to be filtered out. However, this filtering is currently manually designed for the two Mujoco environments being tested. Removing the need for this filtering or developing a more general filtering method will make applying iD-REX to new environments easier.

It is also worth mentioning that similar to traditional IRL methods, iD-REX also requires running RL in an inner loop, making it potentially time-consuming to obtain the final policy iteration. The improvement in expected policy return compared to D-REX should be weighed against the additional computational cost the iterations incur.

Further, we were not able to overcome the apparent inability of the action noise-based approach to iD-REX to explore the policy space and escape being trapped in local reward function maximizers. Modification to the choice of reinforcement learning algorithm, such as implementing Autoregressive Policies (ARP) [9], may help address this shortcoming. ARP can also be used to encourage exploration in the multi-behavioural cloning approach, leading to potentially better results.

Additionally, the results of iD-REX appear highly sensitive to the choice of hyperparameters. Our choice of number of generated trajectories ($\mathcal{N}$) was tuned to achieve a balance between the existing sub-optimal demonstration set and the newly generated demonstrations, and we obtained our results with a single iteration of iD-REX following completion of D-REX ($\mathcal{K} = 2$) to avoid upsetting this balance. Adjustment to the sampling method used when training the reward function should reduce the sensitivity of iD-REX to these hyperparameters and improve robustness.

Another potential area of future work is Conservative Reward Learning. We observed that the learned approximation of the reward function often exhibited features of overconfidence, as seen in Figure 11b. One way to address this might be through ”Conservative Reward Learning”. Similar to Conservative Q-Learning, the idea is to reduce the estimated reward on states not observed by it. Making the reward function updates more incremental may also help with environments with simpler reward functions which iD-REX may ”overshoot” the true reward in fewer iterations.

Finally, we do not quantitatively evaluate the accuracy of our learned reward functions directly. Although the performance of the resulting policies provide some indication as to the correctness of the learned reward functions, a qualitative comparison of the Ground Truth vs Predicted Reward plots indicate that the iD-REX rewards are simply more effective for training Proximal Policy Optimization policies rather than more accurate than those generated by D-REX.

6 Conclusion

We presented iD-REX, an algorithm which extends D-REX by iteratively generating successive sets of demonstrations on the previous iteration, and aggregating those demonstrations to train successively better reward functions and policies. We explored three variants of this algorithm; the first probabilistically interpolates between two learned policies to extend the set of ranked suboptimal demonstrations; the second adds random noise to a learned policy's actions to generate a new set of ranked sub-optimal demonstrations; the third uses learned policies as the source of new behavioral cloning policies which are then degraded with random actions to extend the ranked demonstration set. We evaluated this algorithm in the Mujoco Hopper and Half-Cheetah environments, and found that while the first two variants often fail to outperform D-REX or even the demonstrations, the third outperforms D-REX by 24% and 201% on those environments respectively. These results suggest that the logic of D-REX may be generalized to an iterative process, and that through repeated iterations convergence to a near-optimal policy may be possible.

Acknowledgements

iD-REX was implemented by extensively modifying the original code for D-REX by Brown et al. accompanying [3]. The code for D-REX is publicly available on Github.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[2] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 783–792. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/brown19a.html`.

[3] D. S. Brown, W. Goo, and S. Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 330–359. PMLR, 30 Oct–01 Nov 2020. URL `https://proceedings.mlr.press/v100/brown20a.html`.

[4] B. D. Ziebart, A. Maas, J. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008.

[5] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner. Bayesian optimization for learning gaits under uncertainty. *The International Journal of Robotics Research (IJRR)*, 36 (10):1073–1087, 2017. doi:10.1177/0278364917722396.

[6] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning (ICML)*, volume 48, pages 49–58, 2016. doi:arXiv:1603.00448.

[7] J. Ho and S. Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf`.

[8] D. Sadigh, A. D. Dragan, S. S. Sastry, and S. A. Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017. doi:10.15607/RSS.2017.XIII.053.

[9] D. Korenkevych, A. R. Mahmood, G. Vasan, and J. Bergstra. Autoregressive policies for continuous control deep reinforcement learning. *CoRR*, abs/1903.11524, 2019. URL `http://arxiv.org/abs/1903.11524`.

Appendix: Group Contributions

Jazib Ahmad contributed to the development of the idea of iD-REX and the writing of the project proposal. He also assisted in modifying the original code of D-REX to iD-REX with Probabilistic Interpolation. He provided valuable insights into explanations for some of the observed results. Finally, Ahmad contributed to the preparation of the midterm progress report, final presentation and report.

Geoffrey Harrison was responsible for the initial concept of iD-REX and its variants, as well as for programming and developing their implementations. He also performed the empirical validation of the algorithms, compiled their results, and extensively documented their methodologies and evaluation in the project presentation and report.

Anh Tuan Tran contributed to the initial proposal, midterm progress report, presentation and report. He also contributed to the initial version of the code by modifying the original D-REX implementation.