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

Prabin Kumar Rath prath4@asu.edu
Sok Yan Poon spoon3@asu.edu
Pranit Sehgal pvsehgal@asu.edu
Nilay Yilmaz nyilmaz3@asu.edu

1 Introduction

For successful Human-Robot Collaboration, robots must learn new tasks and work alongside different human workers without requiring extensive reprogramming. A potential solution to the above problem is Learning from demonstration (LfD). It is a paradigm of AI where an agent learns a policy from the sample trajectories demonstrated by another expert agent. Inverse reinforcement learning (IRL) is an LfD approach that optimizes the reward function that can correlate with the expert's behavior. Once an optimal reward function is available, conventional reinforcement learning algorithms can use it to learn a generalized policy. One of the state-of-the-art IRL methods is T-REX (Trajectory-ranked Reward Extrapolation) [1] which depends on ranked demonstrations to learn the underlying intention of the expert agent. A follow-up paper from the team introduces D-REX [2] (Disturbance-based Reward Extrapolation) which uses different levels of noise injection on expert trajectories to generate automatically ranked demonstrations. Both algorithms claim better than demonstrator performance which motivates the use of these algorithms in complex IRL tasks. In this project, we propose three improvements to the baseline D-REX algorithm to improve the stability of the reward learning process. We validate our proposed changes by implementing the D-REX algorithm using open-source libraries and show a 25% reduction in performance variability and a 30% improvement in average rewards over the baseline implementation.

2 Related work

Two popular LfD methodologies are Imitation Learning and Apprenticeship Learning (also known as Inverse Reinforcement Learning, IRL). In robotics, the aim of LfD is to acquire the ideal behavior that is otherwise very complex to program manually. Imitation learning approaches this problem by optimizing a supervised policy that tries to mimic the expert behavior including the suboptimal traits of the assumed expert agent. Thus, vanilla imitation learning algorithms fail to generalize to novel observations in the environment. Several existing works have focused on IRL algorithms to learn reward functions from suboptimal demonstrations. The foundation work that guides our baseline is the human preference learning algorithm Deep Reinforcement Learning from Human Preferences (DRLHP) by Christiano et al. [3]. They proposed an online preference-based optimization method for training reward function and policy simultaneously. Following these ideas, Daniel, et al. proposed the T-REX [1] algorithm that utilizes the Bradley-Terry and Luce-Shephard preference model [4] on ranked demonstrations and learns a reward function that generalizes on observations beyond the suboptimal demonstration data. One of the challenges with T-REX is capturing accurate action labels for ranked demonstrations which are prone is human-error. Addressing this issue Daniel, et al. proposed the D-REX [2] algorithm that first trains a Behavior Cloning [5] policy on expert demonstrations and then adds different levels of noise to it for generating ranked trajectories for T-REX. Both of these algorithms are unique in that they can outperform the demonstrator without additional external knowledge and scale to high-dimensional environments such as Atari games. In contrast, prior work on IRL has relied on assumptions such as the optimality of the demonstrator's policy or the need for additional knowledge such as feature representations. Furthermore, D-REX achieves better performance than state-of-the-art imitation learning and IRL methods [2].

Preprint. Under review.

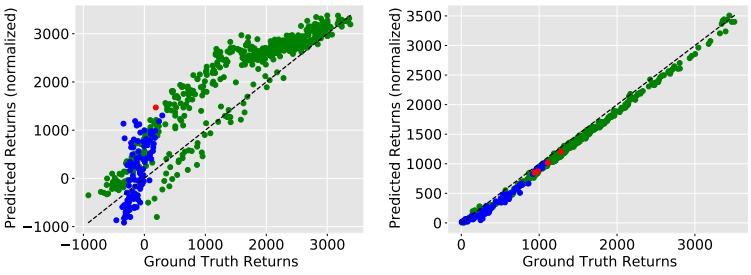

Figure 1: Correlation between ground truth reward returns and D-REX learned reward returns of Half-Cheetah (left) and Hopper (right) respectively.

3 Baseline results

The first step of D-REX is to use behavioral cloning to imitate the expert policy. Different levels of noise are injected into the learned policy π_{bc} and numerous rollouts are performed to generate trajectory data. These trajectories are automatically ranked by preferring trajectories with lower noise over the trajectories with higher noise. These ranked trajectories are then used to recover the reward function using the T-REX algorithm. We present the statistics of D-REX trained on the two default continuous domain MujoCo RL tasks available in OpenAI Gym [6] i.e. Hopper and Half-Cheetah. The expert agents for these environments were obtained by training an RL policy on the ground truth reward function available in Gym. Figure 1 shows the correlation between the returns from the ground truth reward function with the learned D-REX reward function.

Proximal Policy Optimization (PPO) [7] was used to train the policy on the learned reward function. Table 1 presents the result of 15 different runs each on the Hopper and Half-Cheetah environments using the default demonstration data from the D-REX authors. Each run is a 4-step process i.e behavior cloning, noise injection, reward extrapolation, and reinforcement learning. Tensorflow implementation of the algorithm and expert trajectory data were obtained from the D-REX Github archive [8].

Table 1: Returns from PPO policy trained on learned reward function.

Environment	**Demonstration (Avg, Max)**	**D-REX (Avg, Max, Std)**
Hopper	1029, 1167	1307, 2560, 644
Half-Cheetah	187, 187	650, 771, 200

After analyzing the baseline implementation, we notice that the results are inconsistent and the learned policies exhibit significant performance variability. This can be observed from the standard deviation in results reported by authors in the D-REX paper and also from our experiment results in Table 1.

4 Our approach

We propose three enhancements to the baseline approach in order to decrease performance variability and increase the average returns.

Improved Preference Model: The baseline uses the following Luce preference rule shown in Eqn 1

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

The above equation has cumulative sum of rewards as the value for trajectory segments. However, RL literature usually suggests discounted sum of rewards as the value function. Furthermore, the above equation assumes that all the rankings are correct. The DRLHP paper suggests that there is

always a probability of wrong preference and hence they use a preference noise term in their loss function. We integrate these ideas into the baseline's loss function as shown in Eqn 2. Here η is the preference noise probability.

$$
\begin{aligned}
L(\theta) \approx -\frac{1}{|P|} \sum_{(i,j)\in P} \eta * 0.5 + (1-\eta) * log \frac{1}{exp(x)+1} \\
x = clip(\sum_{s(t)\in\tau_i} \gamma^t \widehat{R}_\theta(s) - \sum_{s(t)\in\tau_j} \gamma^t \widehat{R}_\theta(s), -th, th)
\end{aligned}
\tag{2}
$$

The magnitude of the exponential term in the denominator in Eqn 1 depends on the difference between cumulative rewards of the trajectory segments. We have observed that some preference comparisons lead to an excessively large difference, which creates numerical instability and interferes with the backpropagation process during reward optimization. Hence, we clip this exponent to keep it within a threshold. For our experiments, we use $\gamma = 0.99$, $\eta = 0.1$, $th = 50$.

Fixed Horizon Rollouts: For our experiments, we use fixed horizon rollouts of length 1000 steps whereas the baseline uses variable horizon rollouts. The IRL literature suggests that variable horizon rollouts have an inductive bias that can limit the quality of the learned reward function. Fixed horizon rollouts prevent such inductive bias. As per the DRLHP paper [3] fixed horizon rollouts force the reward function to encode information available only from the environment, thus learning a more generalized function for reward extrapolation. Variable horizon rollouts leak information from the true reward function thus biasing the reward network towards the ground truth reward.

Single Reward Network: The baseline uses a reward network ensemble to regularize the learning process. We utilize a single reward network with input normalization. We also scale the output with a $tanh$ activation function. The normalization and scaling helps in regularizing the reward learning process effectively with 3 times fewer parameters.

We analyze the results of our proposed changes with 15 experiments each conducted with a different random seed. Table 2 shows the results comparing the performance of the learned policy before and after the changes.

Table 2: Returns from PPO policy trained on learned reward function.

Environment	**Demonstration**	**D-REX**	**Ours**
	(Avg, Max)	**(Avg, Max, Std)**	**(Avg, Max, Std)**
Hopper	1020, 1508	2072, 3023, 1574	2079, 3307, **940**
Half-Cheetah	711, 844	1584, 1809, 128	**2060**, 2323, 110

The standard deviation of the returns obtained after our changes is lower than the baseline, indicating that the learned policy is more consistent in its performance. For the Half-Cheetah environment, the learned policy outperforms both the demonstration and D-REX in terms of average and maximum returns. Figure 2 shows the correlation between the ground truth reward and the predicted reward for both environments.

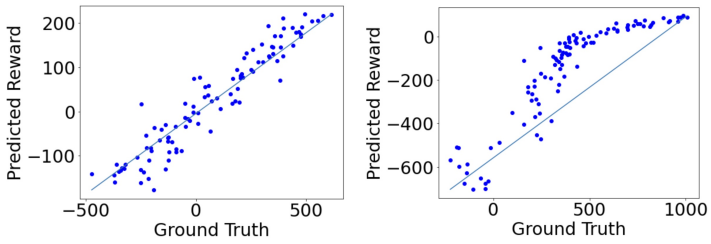

Figure 2: Unscaled correlation between ground truth reward vs our approach predicted reward returns of Half-Cheetah (left) and Hopper (right) respectively.

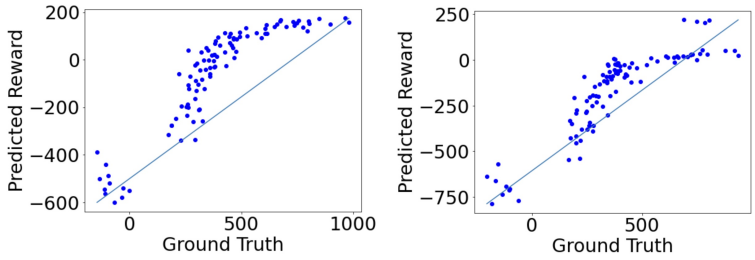

Figure 3: Unscaled correlation between ground truth reward vs our approach predicted reward returns of Hopper with $\eta = 0.05$ (left) and $\eta = 0.2$ (right) respectively.

When we compare Figure 2 to Figure 1, we can see that the changes we made have improved the alignment between the ground truth and predicted rewards for Half-Cheetah. This improvement is reflected in the higher average reward shown in Table 2. On the other hand, for Hopper, we find that the alignment is better with the baseline D-REX algorithm. However, we also note that the average and maximum rewards after our changes are similar or better than the baseline. We suspect that there may be multiple effective reward hypotheses for training the Hopper and that our reward function has learned a different but equally effective function as the ground truth. Figure 3 shows the results of our experiments with two different η values. The misalignment seems to be agnostic of hyperparameter changes for the Hopper environment.

Generating noise-injected trajectories is a crucial part of the project as T-REX learns the reward function by ranking the noisy demonstrations. We studied the effect of different noise types and parameters on the performance of the policy. We experimented with three different types of noise i.e. Epsilon greedy, Normal action, and Ornstein-Uhlenbeck (Brownian motion). The value of epsilon determines the degree of the exploration for Epsilon noise. Normal action, adds noise to the policy output by sampling from a zero mean Gaussian distribution. Ornstein-Uhlenbeck (OU) noise consists of a temporal correlation property that keeps current actions close to the previous actions. Table 3 presents the returns of the PPO policy from our experiments with different noise types. We observed that Epsilon noise works best for Hopper and Half-Cheetah domains.

Table 3: Returns from PPO policy trained on learned reward function with different noise types.

Environment	**Hopper**	**Half-Cheetah**
	(Avg, Std)	**(Avg, Std)**
Epsilon Greedy	2079, 940	2060, 110
Normal Action	676, 456	493, 189
Ornstein-Uhlenbeck (OU)	13, 12	1220, 938

5 Conclusion and Future work

We implemented the D-REX algorithm from scratch using the state-of-the-art tools from stable_baselines3 and imitation packages. Our improvements to the preference model and reward network lead to more stable rewards with an average reward higher than the reported results in the paper. The baseline implementation takes an average of 1 hour to train the reward and learn the policy. Our implementation completes both tasks in under 20 mins. We observed that the baseline D-REX fails to extract a reward function for complex environments such as Ant and Humanoid that involves higher degrees of freedom. Hence, we plan to implement the SSRR [9] algorithm that is built upon D-REX as its baseline. We would compare the effectiveness of our proposed changes on top of SSRR implementation. Our implementation is available at GitHub at Beyond-Demonstration.

References

[1] Brown, Daniel, et al. "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations." International conference on machine learning. PMLR, 2019.

[2] Brown, Daniel S., Wonjoon Goo, and Scott Niekum. "Better-than-demonstrator imitation learning via automatically-ranked demonstrations." Conference on robot learning. PMLR, 2020.

[3] Christiano, Paul F., et al. "Deep reinforcement learning from human preferences." Advances in neural information processing systems 30 (2017).

[4] E.E.M. van Berkum. "Bradley-Terry model". Encyclopedia of Mathematics. Retrieved 18 November 2014.

[5] Torabi, Faraz, Garrett Warnell, and Peter Stone. "Behavioral cloning from observation." arXiv preprint arXiv:1805.01954 (2018).

[6] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[7] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[8] https://github.com/dsbrown1331/CoRL2019-DREX

[9] Chen, Letian, Rohan Paleja, and Matthew Gombolay. "Learning from suboptimal demonstration via self-supervised reward regression." Conference on robot learning. PMLR, 2021.