OUTPERFORMING SUBOPTIMAL DEMONSTRATIONS WITH T-REX AND AIRL

Sahil Arora
College of Computing
Georgia Institute of Technology
Atlanta, GA 30308
sarora64@gatech.edu

Taylor Del Matto
College of Computing
Georgia Institute of Technology
Atlanta, GA 30308
tdelmatt@gatech.edu

Nathan Vaska
College of Computing
Georgia Institute of Technology
Atlanta, GA 30308
vaskas@gatech.edu

ABSTRACT

In this work we attempt to train an inverse reinforcement learning algorithm to learn an optimal reward function from suboptimal demonstrations. We introduce enhancements to an adversarial inverse reinforcement learning (AIRL) framework to handle suboptimal demonstrations with trajectory-ranked reward extrapolation (T-REX). They include iterative T-REX (IT-REX) and adversarially pretrained T-REX (PAT-REX). We compare the performance and sample efficiency of IT-REX on sets of suboptimal demonstrations in a variety of simulated environments relative to AIRL and T-REX baselines.

1 Introduction

Reward functions are a robust and transferable way to define a task and can be used to train policy agents via reinforcement learning. Inverse reinforcement learning algorithms (IRL) are a class of methods that can be used extract a reward function from a set of demonstrations when no environmental reward function is defined. While inverse reinforcement learning has been effective at inferring reward functions from demonstrations, it relies on the assumption that the demonstrations are generated from an optimal policy.

Given a suboptimal set of demonstrations, IRL algorithms will learn correspondingly suboptimal reward functions. Any agents trained on such reward functions are unlikely to achieve optimal performance. Recent advancements in inverse reinforcement learning focus on challenging that assumption by ranking different suboptimal demonstrations to infer an optimal reward function. This project seeks to extend such a method for learning reward functions from suboptimal demonstrations that will outperform the original demonstrations.

Two relevant algorithms to our project are Adverse Inverse Reinforcement Learning (AIRL) [1] and Trajectory-ranked Reward EXtrapolation (T-REX) [2]. AIRL is an GAN based architecture that can be used to successfully recover a policy from demonstration. However, agents trained using AIRL are unlikely to outperform suboptimal demonstrations, since they are trained to mimic the demonstrator policy as closely as possible. T-REX is an algorithm that can be used to learn a reward function that accurately ranks a set of suboptimal demonstrations. The learned reward function naturally encodes information about what makes one trajectory superior to another, and can be used to train an agent from a random initialization to outperform the best suboptimal demonstrations.

2 Related Work

AIRL has two components; a policy generating network and a discriminator. The policy network $\pi(a|s)$ generates trajectories for the task by taking in a state s and producing a probability distribution over the actions a. The discriminator is parameterized by a reward neural net g_θ and a value neural net h_ϕ. $g_\theta(s,a)$ and $h_\phi(s)$ can be combined following Eq. 1 to form the reinforcement learning advantage function $f_{\theta,\phi}(s,a,s')$ where s' is the state that resulted from action a. Given a state, an action a, and the resulting next state, the discriminator advantage function can be used to predict whether a state, action, next state triplet was generated by the policy network or a demonstration. The formula for this prediction is given by Eq. 2, where $\pi(a|s)$ is the probability of the agent taking an action in the given state.

$$f_{\theta,\phi}(s,a,s') = g_\theta(s,a) + \gamma h_\phi(s') - h_\phi(s) \tag{1}$$

$$D_{\theta,\phi}(s,a) = \frac{\exp f_{\theta,\phi}(s,a)}{\exp f_{\theta,\phi}(s,a) - \pi(a|s)} \tag{2}$$

The discriminator network is trained using standard binary logistic regression on a mixture of states and actions from expert demonstrations and policy rollouts from the policy generator. As a byproduct of learning to classify expert versus policy generator inputs, the neural network $g_\theta(s,a)$ will also be trained to provide a reward signal that can be used to effectively update the policy generator to match the expert demonstrations. The generator network uses this reward function as part of a policy loss function, allowing the policy network to learn a policy that minimizes the discriminator's classification accuracy. During the AIRL training cycle, the policy generator and discriminator are alternately trained until they reach convergence. If the network has been trained successfully then at convergence the policy network will produce actions that are indistinguishable or nearly indistinguishable from the actions that the demonstrator would have taken.

Unlike other popular IRL algorithms, AIRL does not assume that the demonstrator will provide an optimal demonstration of the task action. However, since AIRL seeks to mimic the behavior of the provided demonstrations as closely as possible, providing AIRL with suboptimal demonstrations for training data will lead AIRL to learn a suboptimal policy. Several methods have been proposed to explicitly handle and outperform suboptimal demonstrations. One such approach is T-REX. T-REX relies on the assumption that better demonstrations correspond to a higher cumulative reward than worse demonstrations. Specifically, T-REX assumes that the reward from better trajectories will be exponentially higher than that of worse demonstrations.

In cases in which there is no environmental or ground truth reward available to objectively determine superior demonstrations, it can be difficult determine the relative optimality of different demonstrations. T-REX relies on the assumption that human expert can recognize superior demonstrations even if they cannot produce them, and addresses this issue by asking a human expert to provide a ranking of demonstrations. Given this ranking and the set of trajectories, T-REX trains a neural network to predict the probability of one trajectory τ_j being higher ranked trajectory than another trajectory τ_i. It does this via Eq. 3, where $r_\theta(s)$ represents a reward function that provides a higher reward to states from higher ranked trajectories and $\hat{J}_\theta(\tau)$ is the cumulative reward of the trajectory. Using this formulation, $r_\theta(\tau)$ can be trained by performing binary classification and results in a learned reward network can extrapolate beyond the training set of demonstrations to accurately rank novel trajectories. This provides a reward signal that can be used by a reinforcement learning algorithm to outperform the given suboptimal demonstrations.

$$P(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j)) \approx \frac{exp \sum_{s\in\tau_j} \hat{r}_\theta(s)}{exp \sum_{s\in\tau_j} \hat{r}_\theta(s) + exp \sum_{s\in\tau_i} \hat{r}_\theta(s)} \tag{3}$$

Other work has been done on extrapolating from suboptimal demonstrations in an inverse reinforcement learning framework, such as self-supervised reward regression. Flaws in applying preference learning and the Luce-Shepard rule to rank different noised demonstrations have been shown [3]. While we do rely on preference learning with T-REX, our rankings are derived from reward collected by the agent in the environment. They are not a result of injecting noise or assuming any linear relationship in noise levels to optimality.

3 Method

The following section describes our experimental set up for comparing the performance of the AIRL and T-REX algorithms when provided with different levels of suboptimality in demonstration. Additionally, we discuss two extensions for AIRL and T-REX that were compared to the baseline performances for AIRL

3.1 Environment Selection

Three OpenAI Gym environments were chosen for use in this project. Still images of each environment are shown in Fig. 1. The first environment is the classic Control task Pendulum, in which the goal is to balance a single degree of freedom arm atop a pivot. This environment was chosen since it was the simplest control task available from OpenAi Gym. The second environment is the Mujoco task Inverted Pendulum, in which a pendulum is balanced by controlling

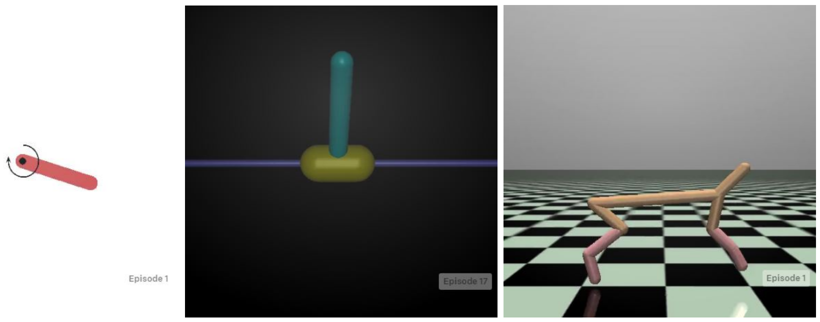

Figure 1: Left to Right: Pendulum Environment, Inverted Pendulum Environment, and Half Cheetah Environment [4]

a cart underneath it. This environment was selected to provide a slightly more challenging environment than pendulum. One drawback of both the pendulum and inverted pendulum environments is that due to their simplicity there is only a small number of possible optimal and suboptimal policies in each environment which might restrict comparisons of algorithm performances. To address this issue, the Mujoco Half Cheetah task was chosen as the final environment. In this task, a multi degree of freedom agent is controlled with the goal of moving forwards as fast as possible.

3.2 Policy Generation and Data Collection

For each environment, the policy network architecture was a two-hidden-layer feed forward neural network that predicts the mean and the standard deviation of a normal distribution for each action in the action space of the agent. The reinforcement learning algorithm Proximal Policy Optimization (PPO) [5] was used with the ground truth environmental reward used to train a set policies for each environment. From each set of policies for each environment, five different policies were chosen as example policies. Each policy corresponded to one of five levels of optimality, starting with a completely random agent and ending with the optimally performing agent. For each policy, a dataset of 250 trajectories were collected to form the training dataset for our baselines and algorithms. Table 1 provides full details about the performance of each selected agent in each environment.

Demonstrator Policy Performance					
Environment	100% (max)	75%	50%	25%	0% (min)
Pendulum	-200	-550	-900	-1250	-1600
Inverted Pendulum	1000	750	500	250	0
Half Cheetah	2250	1500	750	250	-500

Table 1: Level of optimality for selected policies in each environment

3.3 Baselines

3.3.1 AIRL

We utilized two implementations of AIRL for this project. The first implementation was created from scratch based on the original paper. To the best of our knowledge, the only difference between our implementation and the implementation presented in the original paper was our choice to use PPO as our policy optimization algorithm. In the original paper, the reinforcement learning algorithm Trust Region Policy Optimization was utilized instead [6]. Our AIRL implementation was not able to adequately learn the Pendulum task, likely due to insufficient hyperparameter tuning. We adopted a second implementation of AIRL to sanity check performance and generate results on pendulum. The second implementation was a publicly available implementation of AIRL [7]. Both implementations of AIRL had 2 hidden layers for reward, value, and policy networks. Extensive Hyperparameter tuning was required to get the public implementation to achieve adequate performance on Pendulum. In the final actor critic policy network for the public AIRL implementation, both the actor and the critic had two layer, 96 hidden state networks. The reward function was reduced to a one layer 48 hidden state network, and the value function was a two layer 48 hidden state network. With respect to the default public implementation, the final actor critic network dimensions were increased, whereas the final reward and value networks were reduced. Relative to the implementation defaults, the learning rates of the actor, and critic were reduced, and the learning rate of the discriminator was increased. The number of epoch updates for the policy was increased to 80 epochs for every 10 discriminator epochs. AIRL was able to learn an optimal policy for

Pendulum in 2.6 million iterations with these settings. Later on, we also used our own AIRL implementation to sanity check the publicly available implementation when it achieved perfect rewards on random and suboptimal agents.

The AIRL discriminator component was composed of two two-hidden-layer feed forward neural networks with hidden size 32 where one network was considered as the reward network and the other as the value network. The policy network was composed of a two-hidden-layer feed forward neural network. An Adam optimizer with a learning rate of 3e-4 was used for both the AIRL discriminator and generator. Policy rollouts were performed in 16 parrallel environments, with 125 steps being taken in each environment's policy rollout before termination. In each training cycle of the AIRL algorithm, the policy network was trained for 50 epochs and the discriminator network was trained for 10 epochs. In preliminary experiments, we found that the discriminator with these hyperparameters was able to effectively predict whether a state action pair was sampled from a random agent or from the best performing policy agent after 1000 iterations of training. We also found that the policy network with these hyperparameters could be trained with the environmental reward to achieve a high environmental reward for each environment. These preliminary experiments gave us some confidence that we had selected reasonable hyperparameters for AIRL training.

3.3.2 T-REX

We implemented T-REX as inspired by the original paper with a few architectural tweaks to make the architecture more similar to the AIRL architecture. The original implementation consisted of an ensemble of five three-hidden-layer neural networks with hidden layer size of 256, where the different networks received as input 2 partial trajectories of length 50 with state-action information. During policy training, the outputs of each ensemble network were normalized by their standard deviations before being averaged to provide a final reward signal. Each of the ensemble networks received a different partition of the trajectory data set and learned on random combinations of partial pairs trajectory pairs within that partition. The authors of the T-REX paper noted that ensembling was necessary to avoid overfitting the reward signal to limited trajectory data.

We also implemented T-REX with a single three-layer neural network with hidden size 64 that takes as input two state-action pairs from different trajectories. These pairs are labeled with their relative ranking, as the state-action pair from the more optimal trajectory is labeled 1 and the latter is 0. The network is trained to output the corresponding 1s and 0s accordingly with cross-entropy loss taken over the network's logit outputs against the labels. This is equivalent to the authors method of training across partial trajectory pairs where every partial trajectory has a length of one.

This approach is advantageous for our project in that it is learning on the same trajectory scope as AIRL, which also learns on the scale of single state action pairs. Additionally, training on individual state action pairs instead of longer partial trajectories increases the number of pairwise comparisons between trajectories by an order of magnitude at the cost of introducing additional noise into the trajectory ranking. In preliminary algorithm testing in which we compared the ensemble architecture to the single network architecture, we found no difference in performance, likely since the increased amount of data available in the state action training scheme also addresses the issue of overfitting. Since the single network architecture has fewer parameters and is correspondingly faster to train, we chose to use the single head architecture in our experiments.

An Adam optimizer was used for gradient descent on the T-REX network with an initial learning rate of 0.01 with a one cycle learning rate schedule up to 0.1. Minibatches of 1000 state-action trajectory pairs were fed to the model for 5 to 20 epochs. These hyperparameters were not finely tuned, however, we found that they were able to rank trajectories appropriately. This was verified by inputting the state-action pairs from each trajectory to the reward network. The predicted label for each state-action pair in a trajectory were summed together and divided by the number of pairs in the trajectory to calculate an average reward estimate. Across all environments mentioned and their respective policies, trajectories from better policies consistently received higher average reward estimates than those from less optimal or random policies. This gave us confidence that T-REX was properly ranking trajectories and learning a potential extrapolation that can be used for training PPO.

3.4 Iterative T-REX (IT-REX)

As previously described, the original T-REX algorithm can be used to train an agent that outperforms all provided suboptimal demonstrations. Since this is the case, it may be useful to use these outperforming demonstrations to extend the trajectory ranking used as input to the original T-REX algorithm. If additional features of optimality can be feasibly extrapolated from this extended ranking, then the learned policies and their generated trajectories from T-REX may be able to be used to learn an even better reward function by reapplying T-REX. This notion is applied to the algorithm as follows.

We use an agent trained with the T-REX reward signal to generate additional, more optimal trajectories and augment our training data with these trajectories into the ranked set of demonstrations. The T-REX algorithm would then be run again on the augmented ranked trajectory set to generate a new agent that could potentially outperforms all suboptimal demonstrations and the previously learned T-REX agent. Ideally, this process would continue iteratively until the performance of the agent stops increasing. Our implementation consisted of 5 iterations of reward network training, trajectory augmentation, and PPO training.

If the trained agent produced by T-REX is guaranteed to be more optimal than the most optimal trajectory in the training dataset, it could simply be added to the top of the trajectory ranking after each iteration of T-REX. However, it is possible that the policies produced after a single iteration of T-REX training may not be more optimal than the original demonstrator policies. This may be especially likely in our experiments since each iteration of IT-REX was given a fifth of the training epoch as T-REX in order to allow a fair performance comparisons between the two algorithms. If trained agents are incorrectly added to the top of the ranking during each iteration of IT-REX, it is possible that the trajectory ranking would become too noisy to provide an accurate reward signal, causing policies trained on later iterations of IT-REX to perform worse over time. We addressed this by testing two re-ranking strategies that attempt to better locate trajectories within the trajectory ranking after each iteration of T-REX

3.4.1 Naive re-ranking

As explained in the T-REX method section, the average reward estimates were coherent across environments, policy agents, and trajectories of differing levels of suboptimality and could be used to recover the original ranking of the training datasets. In this re-ranking scheme, we generate a score for the new generated trajectories and the training trajectories by feeding their state action pairs to the reward network and summing the cumulative reward. The trajectories are then ranked by the new summed reward estimates.

This strategy is effectively similar to assuming that new trajectories are better if the new policy trained on the learned reward siginal actually outperforms the old trajectories with reference to the learned reward signal. However, we consider this method to be naive as it assumes that the learned reward signal perfectly captures the ranking of the new trajectory in relation to the training trajectories which may not be the case.

3.4.2 Ground truth re-ranking

Instead of trusting T-REX's ability to calculate reward, we can calculate the accumulated reward of the new policy's trajectory and use the ground truth as a basis for ranking. This is similar to using a human to rerank the trajectories between IT-REX iterations. The original ranking of the training set trajectories was generated using calculated environmental reward, so only the environmental reward of the new trajectories needs to be calculated to incorporate the new trajectory into the trajectory ranking. This addresses any posible danger of falsely ranking trajectories that do not properly align with its performance, at the cost of increased expert supervision.

3.5 Pre-trained AIRL with T-REX (PAT-REX)

While AIRL is known not outperform suboptimal demonstrations, it can train agents to perform similarly. T-REX normally starts off with a completely untrained agent, and uses a learned reward signal to train it to outperform the given subtoptimal trajectory ranking. The crux of this algorithm is to train an initial policy to match the behvior of the best suboptimal policy using AIRL, and then to train this policy further with the learned T-REX reward function. By starting from a trained base rather than a random initialization, this algorithm should provide the policy agent should be provided a better chance to outperform the best suboptimal demonstrations.

4 Results

We report the performance and sample efficiency of the PPO agents trained by the respective algorithms described on different levels of suboptimality. Performance is measured as the reward collected by the trained agent, and sample efficiency is measured as the number of epochs the agent had to train for to reach the performance measured. The different levels of suboptimality are tested by taking trajectory data generated from trained agents that collected varying reward.

We set an agent to be at "100%" optimality when it is fully trained and achieves either the maximum reward possible or a standard baseline for reinforcement learning agents. We set "0%" optimality by taking a randomly intialized agent and measured its collected reward. We test the ability of our algorithms on these different levels of suboptimality by feeding it the trajectories up to that optimality, so 100% denotes training with all of the data, including optimal demonstrations, while 50% is training with the random agent trajectories and trajectories generated from policies with 25% and 50% performance. Note that we train AIRL only on the most optimal demonstration at each level of suboptimality.

Some results are not reported due to a myriad of training issues. We could not find a set of hyperparameters to effectively evaluate AIRL on Half-Cheetah environment. We also could not train T-REX effectively on Pendulum either, which is why no results were available for I-TREX. In Inverted Pendulum, AIRL and T-REX acheived similar optimal results but with different sample efficiencies, which were reported. Since we did not have an environment that had useful and meaningfully different results for AIRL and T-REX simultaneously, we could not test PAT-REX, so no results are available for that algorithm.

Pendulum Best Performance				
Algorithm	100%	75%	50%	25%
Ground Truth	-200	-550	-900	-1250
T-REX	-1290	-1452	-1463	-1280
AIRL	**-174.5**	**-870.8**	**-944.5**	**-859.7**

Sample Efficiency (PPO Epochs)				
Algorithm	100%	75%	50%	25%
T-REX	**11000**	**14000**	**15000**	**16000**
AIRL	2660000	110000	545000	635000

Table 2: Pendulum Performance (top) and Sample Efficiency (bottom)

Inverted Pendulum Best Performance				
Algorithm	100%	75%	50%	25%
Ground Truth	1000	750	500	250
T-REX	1000	1000	1000	1000
AIRL	1000	1000	1000	1000
IT-REX Naive	1000	1000	1000	1000
IT-REX Env Rerank	1000	1000	1000	1000

Sample Efficiency (PPO Epochs)				
Algorithm	100%	75%	50%	25%
T-REX	7000	11000	5000	35000
AIRL	37000	27000	30000	35000
IT-REX Naive	**2000**	**3000**	**5000**	**10000**
IT-REX Env Rerank	5000	21000	15000	40000

Table 3: Inverted Pendulum Performance (top) and Sample Efficiency (bottom)

Half-Cheetah Best Performance				
Algorithm	100%	75%	50%	25%
Ground Truth	2250	1500	750	250
T-REX	2248.345	2359.82	2408.672	**35.082**
AIRL	.4	-	-	-
IT-REX Naive	2542.999	**2364.423**	**2513.238**	28.198
IT-REX Env Rerank	**2621.873**	2276.27	2292.04	26.047

Sample Efficiency (PPO Epochs)				
Algorithm	100%	75%	50%	25%
T-REX	67000	232000	230000	157000
AIRL	3015000	-	-	-
IT-REX Naive	**12000**	**66000**	**70000**	**57000**
IT-REX Env Rerank	13000	70000	80000	51000

Table 4: Half Cheetah Performance (top) and Sample Efficiency (bottom)

5 Discussion

5.1 Findings

The highest rewards and lowest PPO epochs for each level of optimality are bolded. While we were unable to report all the results we ideally wanted to, we were able to notice a few interesting trends that we elaborate on below.

5.1.1 IT-REX Performance on Half Cheetah

Our implementation of IT-REX was consistently capable of outperforming the best suboptimal demonstrations in the Half-Cheetah environment, achieving higher performance than the best provided demonstration for tested levels of suboptimality and slightly higher performance than the original, non iterative form of T-REX in a fraction of the number of epochs. This demonstrates that IT-REX, with proper tuning in some environments, is capable of improving the reward signal learned by T-REX by iteratively extending the trajectory ranking. One important point to note is that IT-REX performs approximately as well as T-REX does. If T-REX in it of itself was unable to extrapolate a proper reward function, like in the 25% case, then IT-REX would not be able to as well.

5.1.2 Naive Re-Ranking Outperformed Ground Truth Re-Ranking

In both the Inverted Pendulum and Half-Cheetah environments, IT-TREX received higher reward in less epoch iterations when it re-ranked according to T-REX rewards instead of the reward from the environment. The only time this was not the case was in the lowest optimality case for Half-Cheetah, where reward achieved and epochs ran were negligibly different. We hypothesize that this strategy might be better because it provided a clear reward signal than the ground truth when the extrapolation was effective. The rankings further reinforce the training of the reward network and thus the policy agent, which accelerates training. While the ranking may not be as properly grounded, and could introduce problems if the extrapolated trend is false, this mechanism reaches optimal demonstrations more effectively.

5.1.3 AIRL Outperforms Suboptimal Demonstrations on Inverted Pendulum

Another notable result from our experiments was the performance of the AIRL agent on the Inverted Pendulum task. We found that regardless of the level of optimality of the demonstration provided to AIRL, the trained AIRL agent was capable of achieving the maximum environmental reward. This held true even when we tested AIRL on several completely untrained agents and with both our implementation of AIRL and the publicly available implementation of AIRL. Environmental rewards were recorded for each random agent tested, which confirmed that the random agent performance was universally much lower than the optimally trained agent. Random seeds were also adjusted to ensure that this was not a product of one or two randomly seeded agents generating freak results. Full results for this experimentation are provided below.

Random Agent Performance			
Agent	Environment Reward	Reward - Implementation 1	Reward - Implementation 2
1	7	12	1000
2	2	850	1000
3	8	1000	1000
4	5	1000	1000
5	3	14	1000

Table 5: AIRL Training on Random Agents in Inverted Pendulum

This result strongly conflicted with our expectation that the AIRL agent would be unable to outperform the suboptimal trajectories on which it was trained. For the partially trained but suboptimal agents, our best guess is that within the Inverted Pendulum environment, most demonstrations that achieve any appreciable reward were close enough the optimal trajectory that AIRL could recover a useful reward signal. We are uncertain what, if any, reasons exist to explain the performance of AIRL on the completely untrained agents, which take essentially random actions and achieve negligible reward compared to the optimal behavior. We leave the exploration of this unexpected behavior up to future work.

5.1.4 Poor Extrapolation on Simple Trajectory Spaces

One notable result from our experimentation was the complete failure of the T-REX algorithm to learn an effective reward function on the Pendulum environment. This finding runs counter to our expectation that T-REX algorithm would show the best performance on simpler environments since it would be relatively easy to recognize superior trajectories. In comparison, T-REX manages to achieve optimal performance on the much more complicated Half Cheetah task for every condition except for the extremely difficult 25% optimality condition. These two findings taken together lead us to believe that the poor performance of T-REX is related to the small state and action space of pendulum.

One reason that small state spaces might cause poor performance for T-REX is that the smaller the state and action space, the more overlap there is between states found in lower ranked trajectories and higher ranked trajectories. For example, in the initial state of the Pendulum environment both a random policy and an optimal policy will cause the pendulum to swing back and forth in similar manners, leading to a T-REX network that struggles to differentiate between the two. In more complicated environments with larger state and action spaces like Half Cheetah this overlap is less likely to occur simply because of the larger number of possible state and action combinations.

One way to address this issue would be to use longer trajectory segments for training the T-REX classifier than single state action pairs, since sequences of states and actions are less likely to be shared between higher and lower ranked trajectories than single state action pairs. This is the approach utilized in the original paper, and may be necessary for achieving results in smaller state spaces.

5.2 Limitations

Our project faced several limitations that impacted our ability to collect full results. Our project used a Google Cloud Server instance to access the computational resources required to train our models. The cloud server also allowed us to install Mujoco on a native Linux system rather than an unsupported Windows system. However, we were still limited to training on a single Nvidia T4 GPU.

This restricted the number of trials that could be performed for each model and experiment and limited the amount of hyper parameter tuning that could be achieved. This limitation leaves open the possibility that some of our results were the byproduct of poor hyperparameter tuning. This also resulted in high performance variability with training the reinforcement learning agents across the different environments. This could be due to the brittle nature of the reward function inferred by T-REX or the poor optimization of the PPO hyperparameters. We reported maximal values achieved in our results, but there were many runs that initially performed worse during the first 50000 epochs of PPO training before restarting.

Another limitation was that we were unable to set up screen recording on the the cloud server. This prevented us from directly visualizing the behavior of our agents for the Mujoco environments. Although we were able to get a general sense of their typical behavior from the examples provided by OpenAI Gym, this restriction limited our analysis of agent policy performance to only the accumulated environmental reward.

6 Conclusion

Over the course of this project, we developed two new algorithms for outperforming suboptimal demonstrations and directly compared the existing IRL algorithms AIRL and T-REX. From our experiments, IT-REX was shown to be a more efficient way of learning from suboptimal demonstrations than T-REX, although this was reliant on standalone T-REX being able to extrapolate from trajectories. While we were unable to effectively tune hyperparameters to establish good AIRL baselines or develop PAT-REX since it relies upon AIRL, we still managed to derive meaningful results. We also discovered that on the Inverse Pendulum task, AIRL significantly outperformed completely random demonstrations, which directly conflicts with the conventional understanding of this algorithm's performance.

For future work, testing should be extended to different architectures for T-REX, other policy algorithms such as DDDG, tuning different iterative settings for IT-REX, and reproducing AIRL baselines in more complex environments. If AIRL worked as expected, then PAT-REX would have been a viable architecture to implement and test. Hybrid architectures incorporating preference learning into the adversarial formulation could also be explored for outperforming suboptimal demonstrations consistently. Finally, the causes of the unexpectedly optimal performance of AIRL on the Inverse Pendulum task warrant further exploration and analysis.

References

[1] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[2] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, pages 783–792. PMLR, 2019.

[3] Letian Chen, Rohan Paleja, and Matthew Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. *arXiv preprint arXiv:2010.11723*, 2020.

[4] OpenAI. Openai gym environments. `https://gym.openai.com/envs/#classic_control`. Accessed April 30, 2021.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[6] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[7] Toshiki Watanabe. Gail and airl in pytorch. `https://gym.openai.com/envs/#classic_control`. Accessed April 30, 2021.