Semester Project: Learning from Ranked Demonstrations

Gabriel Hayat
Institute for Machine Learning
ETH Zürich

(Dated: February 29, 2020)

Abstract : Reinforcement learning has recently led to impressive results, especially in the area of Atari games. However, in many applications, rewards are hard to define as to capture all important aspects of a task. This issue is addressed by Inverse Reinforcement Learning (IRL), where an agent's rewards are learnt by observing its behavior. Trajectory-ranked Reward EXtrapolation (T-REX) [1] is an IRL approach that extrapolates beyond a set of (approximately) ranked demonstrations in order to infer high-quality reward functions from a set of potentially poor demonstrations. In this project, the T-REX algorithm is implemented in two Unity environments and its performance under unfavorable conditions is examined.

I. INTRODUCTION

Consider the current case of autonomous driving. The reward function of an autonomous vehicle should capture many different desiderata, including the time to reach a specified goal, safe driving characteristics, etc. As one can conceive, the latter is hard to fully specify. Inverse Reinforcement Learning (IRL) addresses this issue by allowing the agent to learn from demonstrations, transforming the need of specifying the reward function to the task of providing an expert's demonstrations of the desired behaviour.

A major concern about existing IRL methods is their inability to significantly outperform the demonstrator. Trajectory-ranked Reward EXtrapolation (T-REX) [1] is a novel algorithm that allows to significantly outperforms the best demonstration used for training. The latter learns a reward function that explains a ranking of (suboptimal) trajectories induced by pairwise comparisons between them and which leads to functions for which the demonstrated behaviour is not necessarily optimal. The method only requires an initial set of (approximately) ranked demonstrations as input and can learn a better-than-demonstrator policy without any supervision during policy learning.

However, there are doubts as to how well the model performs under unfavourable conditions. This will be investigated with the help of Unity Machine Learning Agents Toolkit (ML-Agents) [2]. ML-Agents is a Unity plugin that enables games and simulations to serve as environments for training intelligent agents. In this project, the T-REX algorithm is implemented in two unity environment: *Gridworld* and *Reacher*. As a first step, the T-REX algorithm is applied to the environments. Secondly, robustness of the algorithm is explored.

II. ENVIRONMENT DESCRIPTION

A. Gridworld environment

The *Gridworld* environment consists of a grid supporting an agent (blue), a goal (green) and a pit (red). The agent's objective is to reach the goal with as few steps as possible, while avoiding the obstacle. This environment is chosen as it is a simple, discrete and well defined Reinforcement Learning task that allows the fair assessment of the T-REX algorithm. The specifications of this environment are listed below:

- Agent Reward Function: -0.01 for every step, +1.0 if the agent navigates to the goal position of the grid, -1.0 if the agent navigates to an obstacle (episode ends)
- Vector Action space: (Discrete) Size of 4, corresponding to movement in cardinal directions.
- Visual Observations: One corresponding to top-down view of GridWorld. (Size (84,84,3))
- Grid size was increased to 10x10 in order to have diverse demonstrations
- Maximum number of steps in an episode is set to 400 steps

Figure 1 shows the setup of the environment.

B. Reacher environment

The *Reacher* environment consists of a double-jointed arm (agent) and a sphere (green) circling around it. The goal of the arm is to get in the area contained by the sphere, and stay in it. This continuous environment is more challenging than the previously described one. This enables the exploration of the robustness of the T-REX algorithm. The specifications of this environment is listed below:

- Agent Reward Function: +0.1 Each step the agent's hand is in the goal location
- Vector Action space: (Continuous) Size of 4, corresponding to torque applicable to two joints.
- Vector Observation space: 33 variables corresponding to position, rotation, velocity, and angular velocities of the two arm Rigidbodies.
- The episode duration is fixed and chosen to be 10 seconds long

Figure 2 shows the setup of the environment.

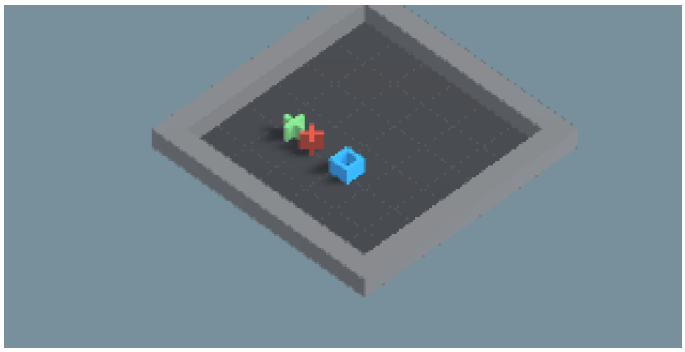

FIG. 1: Gridworld environment set up

FIG. 2: Reacher environment set up

III. TRAJECTORY GENERATION

In the implementation of the T-REX algorithm, the first step is to generate *trajectories*, a sequence of state-action pairs in an episode, in both environments. For the algorithm's benefit, the set of generated trajectories has to be as diverse in quality as possible. This is achieved by training the agent with ML-Agent's implementation of the *Proximal Policy Optimization*((PPO) algorithm and stopping the training process at regular intervals to generate trajectories of increasing reward. The PPO algorithm is a popular Reinforcement Learning algorithm which aims to use a neural network to approximate the ideal function that maps an agent's observations to the best action it can take in a given state.

Name	Symbol	Value (Gridworld)	Value (Reacher)
time horizon	γ	64	1000
batch size	B	32	2024
buffer size	U	256	20240
max steps	E	5×10^6	6×10^6

TABLE I: Parameters for PPO training

The reader is redirected to the original paper [3] for more details about the algorithm. Table III displays the main parameters used during training. Once the trajectories are all generated, the T-REX algorithm further split them into *subtrajectories* of a certain number of steps (environment specific) as a data augmentation technique. We added python scripts to the T-REX package to convert them to a T-REX compatible format (See *GridWorld_demo_converter.py* and *Reacher_demo_converter.py* in provided code). Details regarding the trajectories of each environment are given below.

Gridworld: The aim of this environment is to evaluate the performance of the algorithm under favorable conditions. Thus, a diverse set of trajectories is generated. 20 trajectories from 10 different training stages are generated, for a total of 200 trajectories. As mentioned previously, these are then segmented into subtrajectories of 5 steps. As expected, we can see from figure 3 that the average reward of a trajectory batch increases as the agent training advances. In addition, the average length of a trajectory batch decreases due to the agent reaching the goal (and thus ending the episode) in fewer steps.

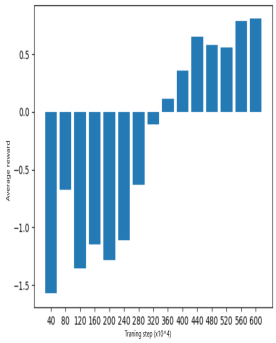

FIG. 3: Average reward of trajectories

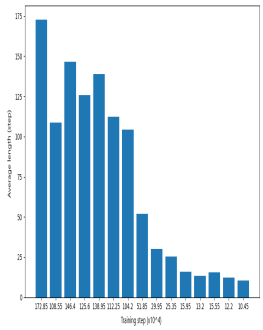

FIG. 4: Average length of trajectories

Reacher: In this setting, we want to evaluate the performance of the T-REX algorithm under unfavorable conditions. We thus generate exclusively low reward trajectories (not exceeding a cumulative reward of 0.149) and examined whether the algorithm still manages to recover the correct behavior. (*Reacher_1* in section VI)

This time, batches of 10 trajectories are generated at 17 different training steps, which are then segmented to subtrajectories of 1 second long.

Furthermore, we later added a trajectory batch of very high quality (3.8 average reward) to examine the effect of this addition on the T-REX model. (*Reacher_2* in section VI). Analysis of the results are made in section VI.

IV. TRAINING THE T-REX MODEL

The original T-REX package contains three important classes for training (see the provided code):

1. *data_loader.py*: This class is responsible of loading and pre-processing the demonstrations into pairs of ranked trajectories to feed to the network. Note that the ranking is the only place where the exact trajectory reward is used, it is ignored for the rest of the training process.

2. *network.py*: This class contains the neural network, which can be viewed as a function $f : \mathcal{T} \to \mathcal{R}$ where $\mathcal{T}$ is the trajectory space and $\mathcal{R}$ is the reward space. This class needed to be re-implemented for each environment.

3. *train_trex.py*: This is the class responsable for training the T-REX network. It draws pairs of ranked trajectories from the generator and feeds them to the network.

Like mentioned previously, each environment needs its own network implementation, due to the semantic difference in the trajectories. Indeed, a *Gridworld* trajectory is described as a sequence of **visual observations** (*vertical dim*, *horizontal dim*, *colour channels*). On the other hand, a *Reacher* trajectory is described as a sequence of **vector observations**, as described in section II. The network architectures for each environment are depicted in the following table. For more details, the reader is redirected to the code.

TABLE II: Gridword and Reacher network architectures

Layer	Output dim
Input	(None, 5, 84, 84, 3)
Conv1	(None, 26, 26, 16)
reLU1	(None, 26, 26, 16)
Conv2	(None, 11, 11, 16)
reLU2	(None, 11, 11, 16)
Conv3	(None, 9, 9, 16)
reLU3	(None, 9, 9, 16)
Conv4	(None, 7, 7, 16)
reLU4	(None, 7, 7, 16)
Flatten	(None, 784)
Dense	(None, 64)
Dense	(None, 1)

Layer	Output dim
Input	(None, 50, 33)
Dense	(None, 16)
Dense	(None, 8)
Dense	(None, 1)

V. TRAINING WITH THE T-REX REWARD FUNCTION

Now that the T-REX model is trained, the next milestone is to re-train each environment using the reward function provided by the T-REX model. Again, this is done using ML-agents' implemention of the PPO algorithm. Instead of using the environment specific rewards given in section II, the reward is determined by running the appropriate network at each step of training.

To achieve this, ML-agents' implementation of the PPO algorithm has to be modified. Again, the modifications required are specific to the environment, thus two classes *rl_trainer_gridworld.py* and *rl_trainer_reacher.py* are added to the ML-agents package. At each step of the PPO algorithm, the visual observation of shape $(None, 1, 84, 84, 3)$ in the *Gridworld* and the vector observation of shape $(None, 1, 33)$ in the *Reacher* setting is fed to the network until the episode ends.

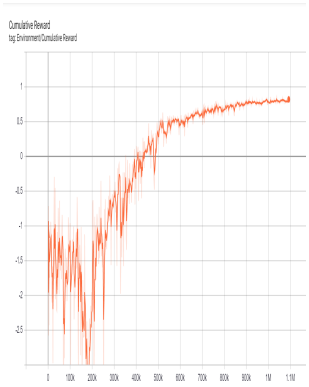

FIG. 5: *Gridworld* average trajectory reward during training

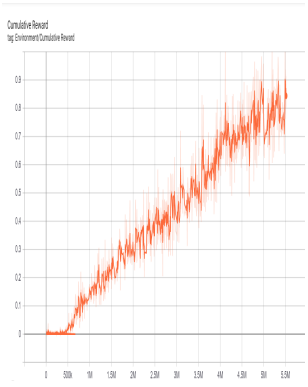

FIG. 6: *Reacher_2* average trajectory reward during training

Figures 5 and 6 show the progression of the cumulative reward along the trajectories during training. In the *Gridworld* case, we can see that the curve starts with a drop indicating that the PPO algorithm initially fails to detect the characteristics of a good trajectory. The curve then starts to rise and flattens out to a reasonable cumulative reward. This indicates that the T-REX model succeeds in inferring a descriptive reward function. In the *Reacher* case, one can see that the curve first stagnates at zero reward, then increases linearly. This indicates that once the agent figures out what makes a good trajectory, it tries to reproduce this behavior for the rest of the training.

VI. RESULTS AND DISCUSSION

Table III displays the results obtained during the experiments. Regarding the *Gridworld* environment, the experiment results are very promising as one can see that the highest reward trajectory generated using the rewards specified in section II has the same value as the one generated using the T-REX rewards. This indicates that the T-REX algorithm successfully manages to extrapolate a the agent's underlying intent (almost) beyond the best demonstration, even when most of the demonstrations are highly suboptimal. Again, this result is achieved when the model uses exclusively the ranking of the demonstrations and not their absolute reward. One can imagine many scenarios where the exact reward value of the demonstrations are hard to compute, but where they can easily be pairwise ranked.

Recall that *Reacher_1* environment is where exclusively poor demonstrations are used to train the T-REX algorithm. In this case, we can see that the model does not produce good results. Indeed, the T-REX model fails to model the task with an expressive reward function, and thus is not able to extrapolate in order to produce higher reward demonstrations. This shows that even though T-REX performs well with a diverse set of demonstrations (*Gridworld* case), the set must still include some high reward trajectories to help the model define this region of the trajectory space.

Recall that *Reacher_2* environment is using the same trajectory set as the previous environment, only adding a batch of high reward trajectories. This is done to measure the degree to which adding a small percentage (17%) of high reward trajectories to a poor trajectory set helps the T-REX algorithm in modeling the reward function. From table III, we can see that this addition increases the highest T-REX reward trajectory value from 0.029 to 0.738. This is a promising result, as it demonstrates that even when the proportion of high quality trajectory among the entire trajectory set is small, the T-REX model still manages to achieve reasonable results. This is an encouraging result as one can easily imagine a situation where the access to high quality demonstrations is sparse.

Env	Highest reward traj	Highest T-REX reward traj
Girdworld	0.795	0.791
Reacher_1	0.149	0.029
Reacher_2	3.785	0.736

TABLE III: The table displays the results found in the experiments. The middle and third columns indicate the highest reward trajectory produced during PPO training when using the environment specific rewards, and the T-REX rewards respectively.

VII. FUTURE WORK

In this project, we have seen that the T-REX model is able to successfully define a reward function of a specific task under favorable conditions. However, the model can be extended in order to further explore its performance.

An interesting idea would be to change the way the reward of a trajectory is computed, so as to give more importance to the states closer to the end of the episode. Throughout this paper, the trajectory reward is defined as $r(\tau) = \sum_{s\in\tau} r(s)$. A coefficient γ_t could be introduced, such that $0 \leq \gamma_t \leq 1$ depending on the state's position in the episode. In the *Gridworld* case, this modification would promote trajectories which end with the agent reaching the goal state.

Another extension of the current model could be to generalize the pairwise trajectory ranking to a ranking of K trajectories. Based on a tuple of K trajectories, the model would aim to output the trajectories in increasing order of cumulative reward. This adds a level of complexity to the task, but would allow the model to extract more semantic differences between the demonstrations.

VIII. CONCLUSION

In this project, we applied the T-REX model to two environments, *Gridworld* and *Reacher*, from the ML-Agents package. We measured the performance of the algorithm in these two environment as well as investigated its performance under unfavourable conditions.

ACKNOWLEDGMENTS

I would like to thank Ivan Ovinnikov and Dr.Luis Haug for their continuous support during this project.

IX. REFERENCE

[1] D. S. Brown, W. Goo, P. Nagarajan, S. Niekum,*Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations* The University of Texas at Austin (2019).

[2] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). Unity: A General Platform for Intelligent Agents.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov ,Proximal Policy Optimization Algorithms, OpenAI (2017).