Learning from Demonstrations: An Alternative to Reward Design

1st RL_H8r
University of Colorado Boulder Smead Aerospace
Boulder, CO USA

I. Introduction

As engineers designing sequential decision-making and control algorithms, we have seen enormous, exciting advances in our ability to solve extremely complicated problems without explicit models. I am, of course, talking about the broad machine learning paradigm of Reinforcement Learning (RL). In RL, we are given an environment and an agent, and the goal of the agent is to optimally interact with the environment to learn an optimal policy, which maps the agent's state in the environment to an action to take in the environment [1]. In the RL paradigm, there is some goal / task encoded in environmental feedback – most often, the task that the agent is supposed to complete is given as a reward signal. This means that as the agent takes actions in the environment, the environment rewards the agent, thus implicitly telling the agent what actions to take. This is not the only way to encode the task for the agent; you could alternatively give the agent the policy and ask them to recover the reward function that caused the policy. With this given "expert" policy and the assumption that the environmental actions can be modeled by a generative MDP, the optimal policy and the reward function both specify the same task [2].

Recently, our interest in RL has greatly expanded due to amazing machine learning technology like deep reinforcement learning algorithms. This newfound enthusiasm is tempered by our the lack of explainability of the Deep Neural Networks (DNNs) representing policy / value estimates, their sample inefficiency, their training instability, and the difficult or impossible process of reward design [3].

One of these problems, the great difficulty of designing reward functions to specify the task at hand, is tackled by Learning from Demonstration (LfD) algorithms [4]. As seen in Fig. 1, the high-level process consists of gathering demonstrations, creating more meaningful representations of those features (i.e. the feature extraction done by all neural networks), learning some representation of the policy from the extracted features, and then refining the policy. In practice, these steps are often combined.

Another of the stated problems with deep RL, its lack of interpretability, could be solved by going with a model-based approach and combining it with formal methods. This would fundamentally change the problem from one of defining an agent-environment simulator and specifying the task as a

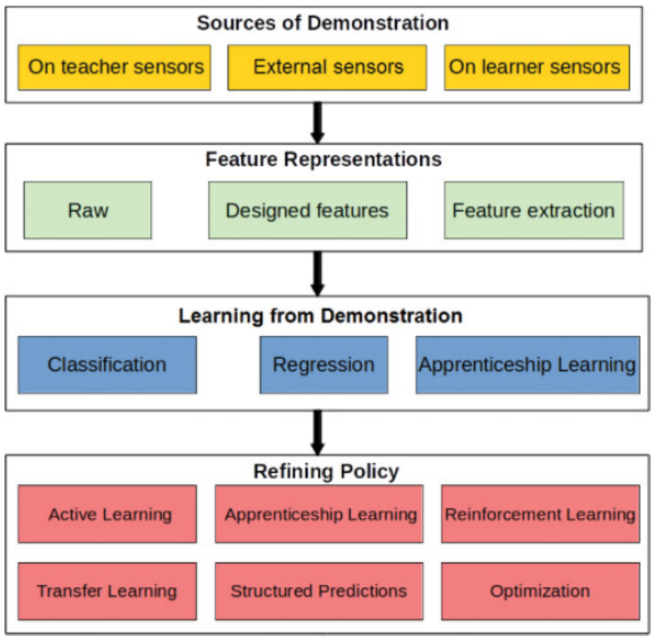

Fig. 1. A flowchart showing the abstract steps and techniques involved in LfD. Reproduced from [1]

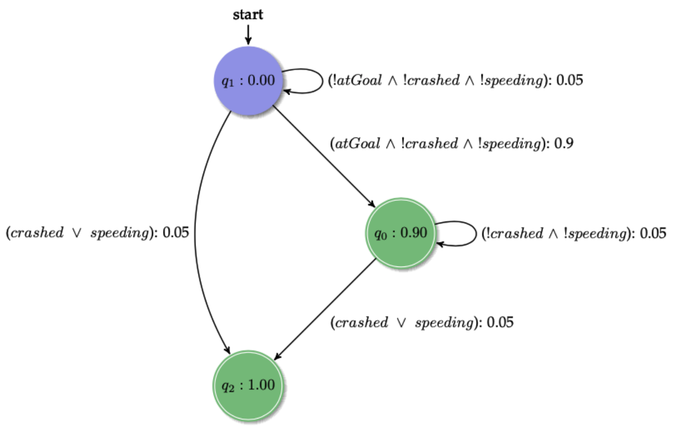
start
$q_1 : 0.00$
$(!atGoal \wedge !crashed \wedge !speeding)$: 0.05
$(atGoal \wedge !crashed \wedge !speeding)$: 0.9
$(crashed \vee speeding)$: 0.05
$q_0 : 0.90$
$(!crashed \wedge !speeding)$: 0.05
$(crashed \vee speeding)$: 0.05
$q_2 : 1.00$

Fig. 2. An example of a probabilistic deterministic finite automaton (PDFA) specification for one of my toy autonomous driving scenarios.

reward function or policy to one of defining a system MDP model and the task specification as a logical structure. This approach, known as formal control synthesis is becoming more and more popular. In his annual review in 2018, Schwarting outlined the progress made in the Verification and Synthesis of decision-making and motion planning algorithms [5], noting that it was of great interest. He noted that its current limitations are mainly its computational expense and the difficulty in formally specifying desired system behavior.

Thus, my research is foces on investigating using machine learning to learn high-level, formal (amenable to formal control synthesis) human specifications for system behavior. More specifically, my research consists of first using probabilistic state-machine learning algorithms (e.g. ALERGIA, MDI [6], or RTI(+) [7]) – a field of machine learning called Grammatical Inference (GI) – to learn an automaton model for human demonstrator's specification for the operation of an autonomous system based on observed symbol sequences (traces) representative of desired system behavior. See Fig. 2 for an example of such a specification. These probabilistic state machines are generative and and when sampled from produce traces in the language of the probabilistic automaton. Then, using tools from formal methods, we compose this specification with a model for the autonomous system's dynamics (e.g. transition system or MDP) to obtain a correct-by-construction policy following the learned specification.

Thus, we have now presented a third way to specify the desired behavior of an uncertain decision-making system. We have seen that we can learn to control an uncertain decision-making system with Learning from Demonstration by learning a policy or reward function, or we could model our system explicitly and formally specify its behavior with an automation representation of the task. As of yet, no one has compared these two bleeding edge methodologies for LfD.

II. Background

As my personal research is in the field of formal synthesis and GI, I would like to focus on more popular, deep learning-based methodolgies from the broad **field of learning from demonstrations (LfD)**, which will make an excellent comparison study for my existing research. There are many, many different ways to approach LfD [4], as can be seen in Fig. 3, but two of the most salient and foundational are Maximum Entropy Deep Inverse Reinforcement Learning (MEDIRL) [8] and Generative Adversarial Imitation Learning (GAIL) [9].

MEDIRL proposes using expert trajectories to recover an estimate for the reward function, which it then uses to run reinforcement learning algorithms on to recover a policy. The interesting thing to note here is that the "maximum entropy" term comes from the need to disambiguate the set of reward functions that match the observed trajectories [8]. In the case of IRL, there are an infinite number of such reward functions, so the novel heuristic here is to choose from those that produce the most entropy. The reward function that induces the most information theoretic policy entropy is likely a good, generalizable reward function as it is the least trivial (from

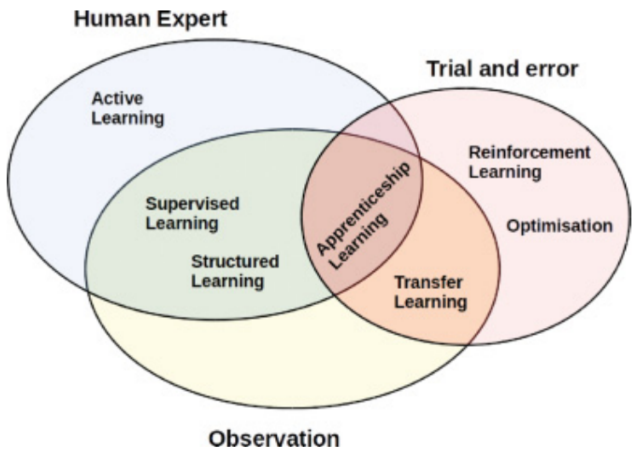

Fig. 3. How all the different forms of LfD are interconnected. Reproduced from [1]

the definition of entropy). This stems from the observation that reward functions that encode complex tasks are rarely trivially definable and sparse, and thus the best choice for a reward function is reward function that applies reward and cost to a large swatch of the state-space [1].

One observation of the observations made in the GAIL paper is that MEDIRL is typically inefficient; most of the time we don't really care about the reward function as only care about learning a controller (policy) for the system. To address this, the authors here suggest a beautiful, if very mathematically intricate, proof that MEDIRL is the dual of the direct occupancy measure matching matching problem (of which they prove all IRL is a subclass of) [9]. The occupancy measure for a policy defined on a finite state / action space is essentially the distribution of state-action pairs that an agent encounters when interacting with the environment according to the policy. Although not explicitly explained, they this assumption they can make this assumption without loss of generality [9].

This occupancy re-framing of the IRL problem leads to the conclusion that the primal to the MEDIRL dual optimization problem is indeed recovering a policy that directly matches the observed expert's problem occupancy measure. With this in mind, the GAIL algorithm is essentially trying to minimize an entropy-constrained occupancy cost regulizer and the entropy of the learned policy. The authors saw the connection between this algorithm and that of generative adversarial neural network architectures (GANs), where a generator network tries to produce examples that fool an adversarial classification network. This unsupervised learning algorithm is exactly the same as the imitation learning algorithm, where the "generator" is the policy network being trained by policy gradient from a reward signal given by the "adversarial" classifier that is trained to distinguish real state-action pairs sampled from the expert from state-action pairs sampled from the policy network [9].

What this amounts to is basically building two neural networks, one that classifies policies as either learned or from the expert, and one standard policy network. The classifier is trained based with a standard classification loss, which is turned into a "reward" signal that the policy network network can use to train with RL. Thus, the policy network never actually interacts with the environment, but you can still use standard policy gradient methods as the basic RL algorithm (e.g. TRPO, PPO2).

The very interesting ideas presented in the GAIL framework are very powerful (GANs are really powerful and cool ways to learn a distribution) make it an appealing methodology for exploration in this project on LfD.

III. Problem Formulation

For this project, we will be working with a labeled gridworld MDP underlying an RL environment. See Fig. 4 for a picture of the proposed gridworld. I am using a labeled gridworld as it will be useful to my work on formal synthesis on the future, and it makes clearer what the task of the robot is, which in this case is **to get to the goal state (green square) while always avoiding the lava squares (orange, textured).**

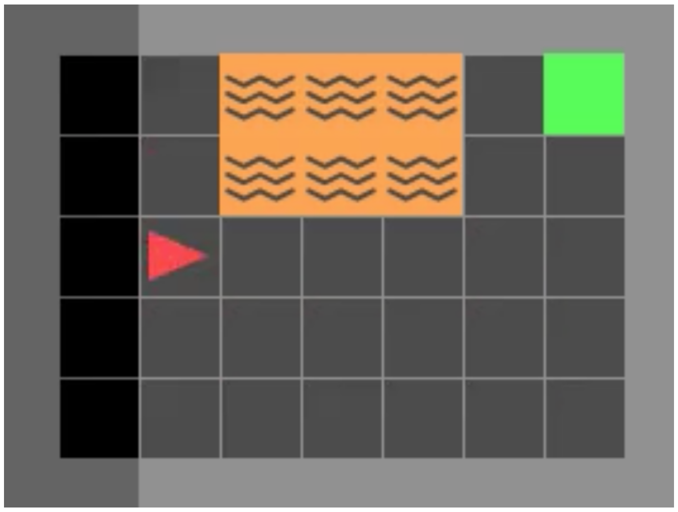

Fig. 4. A custom gridworld environment based on the MiniGW Env. [10]

The gridworld Labeled MDP is defined as a tuple $\langle S, A, T, O, L, R, \gamma \rangle$: [11]

- S is the finite set of states. This 9x7 gridworld has a state space consisting of the grid x and y coordinate, along with the agent's direction.
- A is the finite set of actions. Here, the agent can turn 90° CW, 90° CCW, or go forward one space in the direction it faces.
- $T : S \times A \rightarrow S$ is the state transition function, where $T(s, a, s')$ denotes the probability $P(s'|s, a)$ of reaching state s' from state s by taking action a. Here, the transition uncertainty was simply a 0.05 probability of turning the opposite of the desired direction.
- O is the set of possible state observations. For this problem, we let $O = \{$`empty, atGoal, lava`$\}$.
- $L : S \rightarrow O$ is the labeling function. For our problem, this was used so the agent knew whether the state they were in was in a state labeled `atGoal` or `lava`. Again, this will really only be import for formal synthesis.
- $R : S \times A \rightarrow \mathbb{R}$ is the reward function, where $R(s, a)$ denotes the immediate reward of executing action a in state s, whose absolute value is bounded by $R_{\max}$. Here, we set the reward function to be 0 everywhere except for at the goal (green square), where the agent will receive a reward $r_{goal} = 1 - 0.9 *($`number_of_steps_in_episode`$/100)$. Upon reaching the goal state or the Lava states, the MDP is in a terminal state.
- $\gamma \in [0, 1)$ is the discount factor. Here we set $\gamma = 0.99$

A policy in MDP is defined as a mapping $\pi : S \rightarrow A$, where $\pi(s) = a$ denotes that action a is always executed in state s following the policy π. The value function of policy π at state s is the expected discounted return of starting in state s and executing the policy. The value function can be computed as:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}(s')$$

$$V^{*}(s) = \max_{a} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^{*}(s') \right]$$

It is often useful to express the above equation in terms of Q -function: π is an optimal policy if and only if:

$$\pi(s) \in \underset{a \in A}{\operatorname{argmax}} Q^{\pi}(s, a)$$

where:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^{\pi}(s')$$

Thus, our task was to generate "expert" demonstrations on the grid world environment (MDP defined above) from an expert policy π^{*}_{expert}, and then use a GAIL implementation to learn a policy π^{*}_{GAIL} for the Gridworld without access to a reward signal from the environment itself.

IV. Solution Approach

To solve this problem, I decided to use the stable-baselines implementation of GAIL, as this is by far the most mature and complete RL library out there. Unfortunately, this would turn out to be a bad decision, as it is undocumented that GAIL is no longer well supported by the team, as they are looking to move away from IL in general.

To get an expert policy π^{*}_{expert} on any basic grid-world environment similar to the one I defined in the previous section, I decided to use and tune the stable-baselines PPO2 algorithm to be the expert [12]. To do this, optimized its hyperparameters, resulting in the parameters found in Tables I - III.

From this, I could then generate 100 trajectories from π^{*}_{expert} using rollout simulations (while recording state and

TABLE I
FINAL PPO2 HYPERPARAMETERS PT 1.

cliprange	ent_coef	gamma	lam	learning_rate
0.2	0	0.99	0.95	0.00015

TABLE II
FINAL PPO2 HYPERPARAMETERS PT 2.

n_steps	n_timesteps	nminibatches	noptepochs
128	200000	32	10

action pairs for each episode) and store them for the training process in GAIL.

Then, using the expert trajectory dataset, trained a GAIL learner using the ”optimized” (see results section for discussion on this) hyperparameters given in Tables IV - VI.

From this, I could evaluate the learner's ability to gather reward on the original gridworld environment using sufficient (500) Monte Carlo policy evaluations.

V. RESULTS

All experiments can be reproduced using the `GAIL_testing.ipynb` notebook in my github repo. To run it, you will need to install docker, then build my docker image using the Makefile / Dockerfile I wrote, and then run it using the provided bash scripts. This will open a jupyter notebook and tensorboard in the container and handle the networking between the host and the container. It has been tested to work on both macOS and linux, on machines both with and without GPUs.

To see gifs of both the expert and GAIL policy in the gridworld, you can visit the main page of the github repo.

The results of the learning process are presented here. Shown in Figs. 5 and 6 are the episodic reward and learning curve for the PPO2 RL expert training process. As is quite evident, the PPO2 agent trained well and quickly, needing only 200k environmental interations total to learn a nearly perfect policy. The expert policy had an approximate mean reward of 0.97 $\pm$ 0.01 – a nearly perfect result – when simulated after learning. Looking at the videos of the agent's interaction, the PPO2 expert was never observed to do anything besides quickly and efficiently reach the goal safely.

The GAIL training results were, on the other hand, extremely frustrating. Despite weeks of debugging and working with the package maintainers, it was discovered that the GAIL learner has some pernicious bugs with the handling of the MPI based training environment parallelization that have not been squashed. Thus, as can be seen in Figs. 7 - 9, GAIL never ended up learning the proper thing. As can be seen specifically in Figs 8 and 9, the GAIL training was obviously having problems even with the ”best” hyperparameters chosen. The dicriminator loss never reached 0, even after 1M iterations, and the policy gradient reward signal stagnated over time instead of improving, despite the fact the TRPO RL engine in GAIL should guarantee monotonic improvements in the solution quality. GAIL never received more than 0 reward.

When the expert was moved to a new domain, shown in Fig. 10, its policy was brittle and it did not generalize at all.

TABLE III
FINAL PPO2 HYPERPARAMETERS PT 3.

policy	eval_freq
MlpPolicy	500

TABLE IV
FINAL GAIL HYPERPARAMETERS PT 1.

policy	n_timesteps	eval_freq
MlpPolicy	1e6	1e4

TABLE V
FINAL GAIL HYPERPARAMETERS PT 2.

hidden_size_adversary	adversary_entcoeff	d_step
100	1e-3	10

TABLE VI
FINAL GAIL HYPERPARAMETERS PT 3.

d_stepsize	normalize	gamma
1e-4	False	0.99

VI. CONCLUSION

As can be seen in the Results section, this experiment with GAIL went worse than expected. While eventually I had a very functional and configurable experimentation environment in jupyter, I was ultimately unable to overcome the technical difficulties associated with the GAIL implementation to make it work. Here, we did not accomplish nearly as much as was desired owing both to Imitation learning algorithm's notorious training difficulty, as well as numerous technical problems with the stable-baselines imitation learner, as well as the interactions between gym-minigrid and stable-baselines.

It should be noted that the idea to use PPO2 to learn the expert policy worked exceptionally well, but due to the limited state representation that I used here, it was not able to generalize to new, similar environments.

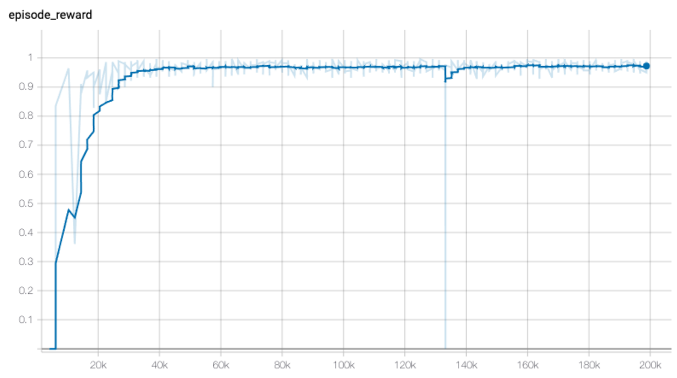

Fig. 5. The final PPO2 training episodic, non-discounted reward as a function of training step.

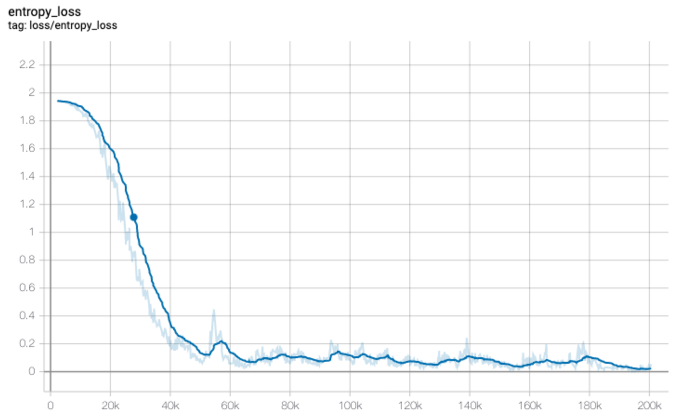

Fig. 6. The final PPO2 entropy loss as a function of training step.

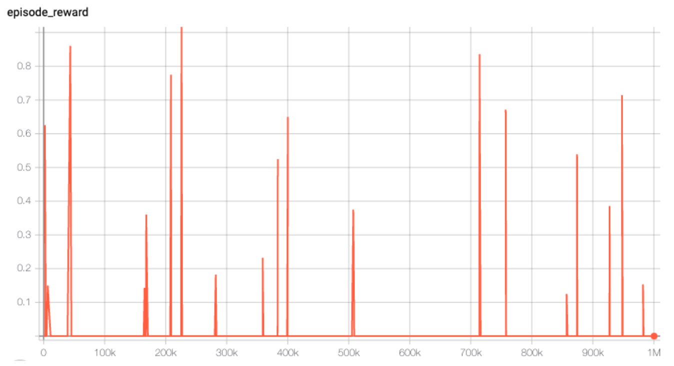

Fig. 7. The final GAIL episodic, non-discounted reward as a function of training step.

Additionally, one thing you will notice is that in the final version of the GAIL policy video on my github page is that GAIL seems to have learned to move around the environment and approach the goal before leaving and making a circle, always avoiding lava. Thus, my hunch is that there is still some sort of bug with the handling of the terminal states of the MDP, causing the demonstration data manager to mangle the final state of the demonstration, such that the GAIL learner

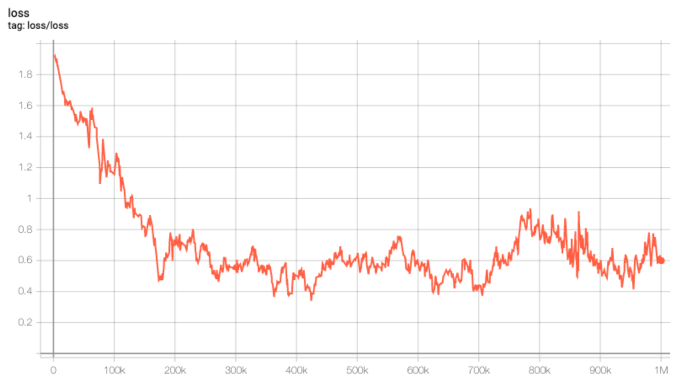

Fig. 8. The final GAIL discriminator classification loss as a function of training step.

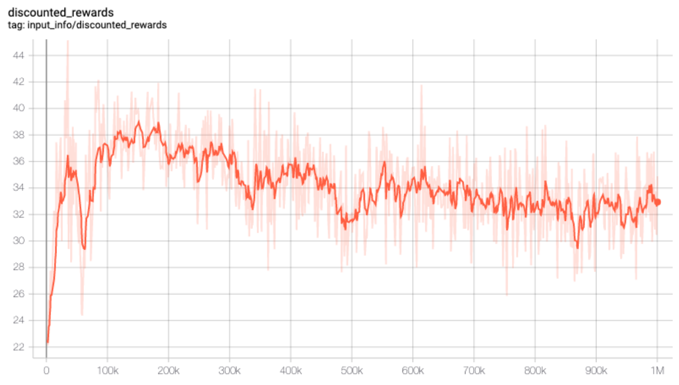

Fig. 9. The final GAIL policy network discounted "reward" signal from the descriminator as a function of training step.

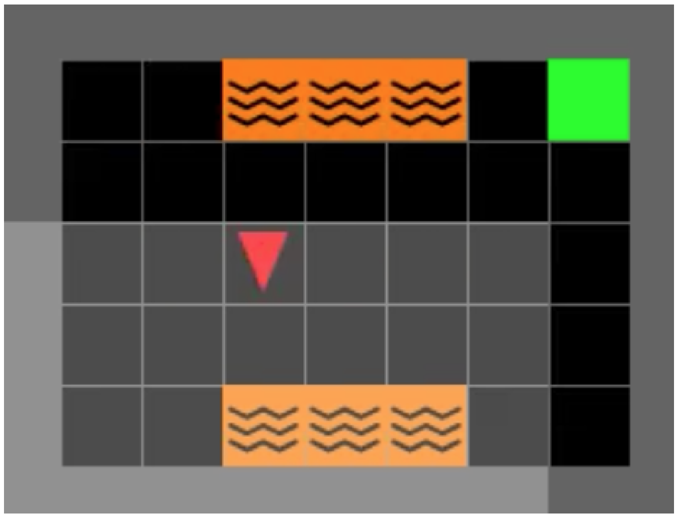

Fig. 10. The final PPO2 policy network trying (and failing) to adapt to a slightly different environment.

is implictly "told" to not enter the goal state.

Regardless, this project was successful in the sense that there is now, for the first time, a connection between the major RL libraries to study task-based LfD in a (potentially partially observable) gridworld environment. I am sure that the technical difficulties can be ironed out with some time off away from the problem – check my github out later for updated results!

References

[1] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," pp. 1–35, apr 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3054912

[2] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 663–670.

[3] A. Irpan, "Deep Reinforcement Learning Doesn't Work Yet," 2018. [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html

[4] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent Advances in Robot Learning from Demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, may 2020.

[5] W. Schwarting, J. Alonso-Mora, and D. Rus, “Planning and decision-making for autonomous vehicles,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018. [Online]. Available: https://doi.org/10.1146/annurev-control-060117-105157

[6] C. de la Higuera, *State Merging Algorithms*. New York, NY, USA: Cambridge University Press, 2013, no. 2004, ch. 16.3, pp. 333–339.

[7] S. Verwer, R. Eyraud, and C. Higuera, “Pautomac: A probabilistic automata and hidden markov models learning competition,” *Mach. Learn.*, vol. 96, no. 1-2, pp. 129–154, Jul. 2014. [Online]. Available: https://doi.org/10.1007/s10994-013-5409-9

[8] M. Wulfmeier, P. Ondruska, and I. Posner, “Maximum Entropy Deep Inverse Reinforcement Learning,” in *ArXiv*, 2015.

[9] J. Ho and S. Ermon, “Generative Adversarial Imitation Learning,” in *NIPS*, 2016.

[10] M. Chevalier-Boisvert, L. Willems, and S. Pal, “Minimalistic grid-world environment for openai gym,” https://github.com/maximecb/gym-minigrid, 2018.

[11] M. J. Kochenderfer, C. Amato, G. Chowdhary, J. P. How, H. J. D. Reynolds, J. R. Thornton, P. A. Torres-Carrasquillo, N. K. Üre, and J. Vian, *Decision Making Under Uncertainty: Theory and Application*, 1st ed. The MIT Press, 2015.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *ArXiv*, vol. abs/1707.06347, 2017.