Inverse Reinforcement Learning via Ranked and Failed Demonstrations

Prabhat Nagarajan
prabhatnagarajan@utexas.edu

Abstract— **In many robotics applications, applying *reinforcement learning* (RL) can be especially difficult, as it depends on the prespecification of a reward function over the environment's states, which is often hard to define. *Inverse Reinforcement Learning* (IRL) [1] attempts to address this problem, by utilizing human demonstrations to learn the reward function, without having a human explicitly define it. However, IRL has the underlying assumption that the human demonstrator performs optimally, which can be an unrealistic assumption in extremely complex tasks. Attempts to mitigate this optimal demonstration assumption have been proposed, such as active learning and interactive corrections, but these methods often assume that task execution can be paused, or that the demonstrator can provide the optimal action when queried. With the goal of developing an IRL framework that can perform as successfully as the current state-of-the-art algorithms in reinforcement learning, we introduce a new algorithm, *IRLFR*, that combines inverse reinforcement learning from failure (IRLF) and inverse reinforcement learning from ranked demonstrations to create a more general framework that can address difficult problems in robot learning with suboptimal demonstrations, so that robots can perform at a superhuman level while minimizing the need for interactive corrections and active learning. We validate our approach in a gridworld domain, where it is nonintuitive for humans to play optimally, and where existing reinforcement learning algorithms can easily find the optimal policy. Our results show that IRLR and IRLFR are frameworks that on average, perform better than the MEIRL baseline performance.**

I. Introduction

In recent years, Reinforcement learning has had huge successes, particularly in Atari game play [12], [4] and in the game of Go, where Google Deepmind's *AlphaGo* defeated a champion Go player, a feat thought to be decades out. In 2015, Google Deepmind created *DQN* [12], a general-purpose deep reinforcement learning algorithm that learns to play at an expert level on a variety of Atari games, such as Breakout, Space Invaders, and more. However, in all of these domains, the researchers did not experience the pervasive problem in reinforcement learning of defining a reward function, since the domains had natural easy-to-define reward functions. However, in many robotics applications, the reward function for a task is often not straightforward. *Inverse Reinforcement Learning* [1] aims to address the difficulty of explicitly defining a reward function by using human demonstrations to learn the environment's reward function. This learned reward function is then used in applying reinforcement learning algorithms to find the optimal policy.

Our primary motivation is to extend inverse reinforcement learning to develop a truly general IRL framework that creates autonomous agents that can solve extremely complex tasks where standard reinforcement learning fails, tasks where the reward function is unclear. Specifically, we believe that a general framework has the following properties:

1) **Model-Free** A truly general framework yields agents that do not need previous knowledge of the transition dynamics of the system. For example, Google Deepmind's Atari game-playing agent learns fully autonomously, without any previous knowledge of the games that it plays.
2) **Efficient Learning with Suboptimal Demonstrations** The average human cannot provide perfect play of games like Space Invaders. Similarly, in many tasks, it is unrealistic for the human to demonstrate the optimal actions.
3) **Efficient in Large State Spaces** In practice, most tasks have large state spaces. That is, every state cannot be stored within a computer.
4) **Minimal Human Intervention** In many tasks, interactive corrections are not possible, especially when it is not possible to pause task execution. Additionally, a general framework should not be susceptible to issues that arise in active learning, such as ill-posed queries, or situations where the human cannot identify the optimal action.
5) **Feature Autogeneration** A fully autonomous framework should not need hand specified features provided by humans. We do not focus on this, but there has been work in this area [5].

In this work, we attempt to address the second point, learning from suboptimal demonstrations. To create a framework with this properties, we must extract as much relevant information from the initial demonstrations to assist in learning the reward function. If we cannot expect optimal demonstrations all of the time, we should try and leverage the differences between demonstrations. Intuitively, if we understand what makes some demonstrations better than other demonstrations, it improves our understanding of what an optimal demonstration entails. In particular, we focus on:

1) **Failed Demonstrations** Humans are often unsuccessful in their demonstrations. These failed demonstrations can provide useful information to the agent about what *not to do*. One can see how this can be useful, especially when perfect, optimal demonstrations are absent.
2) **Ranked Demonstrations** It is difficult for humans to provide optimal demonstrations, as well as to provide

demonstrations of a specific targeted quality. However, it is often intuitive for humans to determine whether one execution of the task is preferred to another. Ranking the human's demonstrations can provide additional information used to learn the reward function.

In order to demonstrate the effectiveness of our framework, we train an agent to navigate in a 7x7 gridworld domain. We choose this domain because it is simple to provide suboptimal and failed demonstrations, and it is straightforward to measure the trained agent's performance.

The main contributions of this paper are:

1) To our knowledge, we are the first to create an IRL algorithm that exploits preferences by modifying the weights when computing the empirical feature expectations (see section III). We do this through our algorithm, IRLR.
2) To our knowledge, we are the first to combine both failed demonstrations and ranked demonstrations into a coherent framework. We do this through our algorithm IRLF.
3) To our knowledge, we are the first to successfully have better performance through ranked demonstrations compared to unranked demonstrations.

The remainder of this paper is structured as follows. We first discuss prior work in IRL and learning from failed demonstrations, ranked demonstrations, and interaction. We then describe the problem formulation and provide relevant background. We then introduce our algorithms, followed by our experiments and results.

II. Related Work

There has been work in learning from failed demonstrations within the Learning from Demonstration (LfD) community, as in [14], [10], [9], which involve the use of a "Donut distribution" and Donut Mixture Models (DMMs), neither of which are used within an IRL framework.

Within IRL, there has not been much work on Learning from failed demonstrations, except for recent work [15], which is the method we use within our framework. This method, called Inverse Reinforcement Learning through Failure (IRLF) works by collecting both successful demonstrations and failed demonstrations, and is defined within the Maximum Entropy IRL framework [17], by learning a reward function based on computed weights for both the failed demonstrations and the successful demonstrations. More details regarding these algorithms will follow in subsequent sections.

Additionally, there has been work within the LfD community on leveraging rankings between demonstrations to find an optimal policy. These works [2], [11], [13], [3], and much of the other literature on leveraging rankings often involves active learning, or is not applied to an inverse reinforcement learning setting.

Within IRL, there has been some work in learning from rankings [16], [8], [7]. [8] utilizes a scoring of demonstrations, differing from our work in that we rank demonstrations relative to one another, without imposing a score on it. [7] is restricted to the domain of natural language processing, whereas ours is more general. Most closely related to our framework is [16], which weights the input demonstrations by their ranking. However, in their work, [16] were unable to obtain improvements over standard IRL without ranked demonstrations. Their approach is very similar to ours, however our approach differs in that the weights we place on input demonstrations are different, and we combine learning from ranked demonstrations with learning from failed demonstrations.

III. Methodology

A. Preliminaries

In this framework, and in most inverse reinforcement learning frameworks, the task at hand is formulated as a *Markov Decision Process* (MDP). An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the set of environmental states, and $\mathcal{A}$ is the set of actions available to the agent. In an MDP, an agent is in a state $s \in S$, takes an action $a \in A$, and then transitions to a state $s^{'} \in S$. T is a transition function $T : S \times A \times S \rightarrow [0, 1]$, a mapping from $(s, a, s^{'}) \in S \times A \times S$ to the probability of transitioning to state $s^{'}$ from state s by taking action a. R a *reward function*, a mapping $R : \mathcal{S} \rightarrow \mathcal{A}$, mapping a state $s \in S$ to the reward that the agent receives for reaching s. In IRL, demonstrations are used to learn a reward function, as opposed to assuming a prespecified reward function. With this in mind, an MDP$\backslash R$ is defined as an MDP without a reward function. In an MDP setting, the goal of the agent is to learn the optimal *policy*, where a policy π is defined as $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. That is, $\pi(s, a) = P(a|s)$, the probability that the agent takes action a in state s. The *optimal policy* is the policy π^* that maximizes the agent's expected reward over some number of decisions h [15]. This expected reward given some policy π is formalized with a value function V^{π}:

$$V^{\pi} = E\{\sum_{t=1}^{h} R(s_t, a_t)|s_1 = s\}. \quad (1)$$

It is assumed that the reward function can be expressed as a weighted sum of K feature functions ϕ_k:

$$R(s) = \sum_{k=1}^{K} w_k \phi_k(s), \quad (2)$$

where $w = [w_1, w_2, ..., w_K]^T$ is a weight vector that is to be learned through IRL. The learning agent assumes that it has as input a set of successful demonstrations $\mathcal{D} = \{\tau_d^1, \tau_d^2, ..., \tau_d^N\}$, and a set of failed demonstrations $\mathcal{F} = \{\tau_f^1, \tau_f^2, ..., \tau_f^M\}$. Given these two sets, the *empirical feature expectation* for $\mathcal{D}$ and $\mathcal{F}$ is defined as [15]:

$$\tilde{\mu}_k^{\mathcal{D}} = \frac{1}{N} \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{h} \phi_k((s_t^{\tau}, a_t^{\tau})), \quad (3)$$

where k is some feature. Similar to the empirical feature expectations, we can compute the feature expectations of a policy π. Given a policy π and the set of successful demonstrations $\mathcal{D}$, we can use equations $4-6$ in Algorithm 1 of [17], for all states $s_i \in S$, we can obtain D_s, the expected "state visitation frequency" of state s, under policy π. We can then define:

$$\mu_k^\pi|_\mathcal{D} = \sum_{s \in S} D_s \phi_k(s), \tag{4}$$

the feature expectations under policy π, as in Eq. (6) in [17]. Similarly, we can compute $\mu_k^\pi|_\mathcal{F}$, using the failed demonstrations.

The idea behind IRL is to find a policy whose feature expectations "matches" the empirical feature expectation. That is, to find a policy π such that

$$\mu^\pi|_\mathcal{D} = \tilde{\mu}^\mathcal{D}. \tag{5}$$

The general approach to this is to begin with some initial weights w, compute a policy π using w, and then to update w with gradient descent to reduce $\mu^\pi|_\mathcal{D} - \tilde{\mu}^\mathcal{D}$. In the following subsections, we delve into two algorithms, MEIRL and IRLF, before introducing our algorithms IRLR and IRLFR.

B. Maximum Entropy Inverse Reinforcement Learning (MEIRL)

While we introduce the high level components of MEIRL here, we do not examine MEIRL deeply. For a full treatment, read [17]. Algorithm 1 in [17] provides a more detailed description of the algorithm.

There are three main components to the algorithm: **computePolicy** (as in $1-3$ of Algorithm 1 in [17]), **computeFE**, and **updateWeights**. They are as follows:

Algorithm 1 computePolicy($\mathcal{S}$, $\mathcal{A}$, T, w)

```
1: Set $Z_{s_{terminal}} = 1$.
2: Recursively compute for $N$ iterations.
       $Z_{a_{i,j}} = \sum_k T(s_k|s_i, a_{i,j}) e^{R(s_i|w)} Z_{s_k}$
       $Z_{s_i} = \sum_{a_{i,j}} + \mathbf{1}_{\{s_i = terminal\}}$
3: $P(a_{i,j}|s_i) = \frac{Z_{a_{i,j}}}{Z_{s_i}}$
4: return $P(a|s)$ (the policy)
```

Algorithm 2 computeFE(T, π, $startDist$)

```
1: compute $\mu_k^\pi|_\mathcal{D}$ using Eq. 4.
2: return $\mu_k^\pi|_\mathcal{D}$
```

The function **updateWeights** takes in a step-size α

Algorithm 3 updateWeights(α, w)

```
1: $w \leftarrow w - \alpha(\mu_k^\pi|_\mathcal{D} - \tilde{\mu}_k^\mathcal{D})$
```

Combining all these three steps, we obtain MEIRL.

Algorithm 4 MEIRL(α) ([17])

```
1: while not converged:
2:     $\pi$ = computePolicy($\mathcal{S}$, $\mathcal{A}$, $T$, $w$)
3:     $\mu_k^\pi|_\mathcal{D}$ = computeFE($T$, $\pi$, $startDist$)
4:     $w$ = updateWeights($\alpha$, $w$)
```

C. Inverse Reinforcement Learning from Failure (IRLF)

For a more comprehensive treatment of IRLF, see [15]. To summarize, this method uses two classes of demonstrations: failed demonstrations and successful demonstrations. This modifies MEIRL to incorporate these failed demonstrations. The algorithm works as follows. It computes two feature expectation vectors, one computed from $\mathcal{D}$, and the other from $\mathcal{F}$. It computes two initial state distributions from the demonstrations. That is, it uses $\mathcal{D}$ to compute a distribution over initial states for successful demonstrations, and uses $\mathcal{F}$ to compute a distribution over initial states for failed demonstrations. It then initializes two vectors of weights, $w^\mathcal{D}$ and $w^\mathcal{F}$. It then begins a loop that repeats until convergence. The loop does the following. It initializes the reward function R as the dot product of the feature vector ϕ and $(w^\mathcal{D} + w^\mathcal{F})$. It computes the policy as before, then computes the features expectations of the policy for both the failed and successful demonstrations. It then updates the weight vector $w^\mathcal{D}$ using **updateWeights** defined in the previous subsection. While $w^\mathcal{D}$ is updated through gradient descent ($(\mu_k^\pi|_\mathcal{D} - \tilde{\mu}_k^\mathcal{D})$ is the gradient, see [17] for further details) the minimum solution for $w^\mathcal{D}$ can be obtained analytically (see [15]). Thus we set

$$w^\mathcal{F} = \frac{\mu_k^\pi|_\mathcal{F} - \tilde{\mu}_k^\mathcal{F}}{\lambda}, \tag{6}$$

where λ is a decaying constant. Because the feature expectations of π are computed using both $w^\mathcal{D}$ and $w^\mathcal{F}$, it may affect performance, as $w^\mathcal{D}$ may change slowly, while $w^\mathcal{F}$ may change by large amounts, given that is has an analytical solution. Thus, we begin with a large λ, so that the failed demonstrations initially have little affect, and we then incrementally reduce λ with each iteration of the loop. Formally, the algorithm, as in [15] is the following:

Algorithm 5 IRLF($\mathcal{S}$, $\mathcal{A}$, T, ϕ, $\mathcal{D}$, $\mathcal{F}$, α, α_λ, λ, λ_{min})([17])

```
1: $\tilde{\mu}_k^{\mathcal{D}} \leftarrow \frac{1}{N}\sum_{\tau\in\mathcal{D}}\sum_{t=1}^{h}\phi_k((s_t^\tau, a_t^\tau))$
2: $\tilde{\mu}_k^{\mathcal{F}} \leftarrow \frac{1}{N}\sum_{\tau\in\mathcal{F}}\sum_{t=1}^{h}\phi_k((s_t^\tau, a_t^\tau))$
3: $P_{\mathcal{D}}^{s_1} \leftarrow$ **initialStateDistribution($\mathcal{D}$)**
4: $P_{\mathcal{F}}^{s_1} \leftarrow$ **initialStateDistribution($\mathcal{F}$)**
5: $w_k^F \leftarrow 0$, $\forall k \in \{1, .., K\}$
6: Initialize $w^{\mathcal{D}}$ randomly
7: repeat
8:     $R(s, a) \leftarrow (w^{\mathcal{D}} + w^{\mathcal{F}})^T\phi(s)$, $\forall s \in S$
9:     $\pi \leftarrow$ computePolicy($\mathcal{S}$, $\mathcal{A}$, $T$, $w$)
10:    $\mu_k^\pi|_{\mathcal{D}}$ = computeFE($T$, $\pi$, $P_{\mathcal{D}}^{s_1}$)
11:    $\mu_k^\pi|_{\mathcal{F}}$ = computeFE($T$, $\pi$, $P_{\mathcal{F}}^{s_1}$)
12:    $w^{\mathcal{D}}$ = computeGradient($\alpha$, $w$)
13:    $w^{\mathcal{F}} = \frac{\mu_k^\pi|_{\mathcal{F}} - \tilde{\mu}_k^{\mathcal{F}}}{\lambda}$
14:    if $\lambda > \lambda_{min}$ then
15:        $\lambda \leftarrow \alpha_\lambda \lambda$
16:    end if
17: until convergence
18: return $R$, $\pi$
```

D. Inverse Reinforcement Learning through Failure and Ranking

Our methodology to perform inverse reinforcement learning through rankings is to rank and order the successful demonstrations in order of their successfulness, rank and order the failed demonstrations in order of their unsuccessfulness, and use the demonstration's position in the sorted list to modify its weight in computing the empirical feature count. First, we sort the demonstrations $\mathcal{D}$ such that after sorting, $\mathcal{D} = \{\tau_1, \tau_2, ...\tau_N\}$, and if $i \geq j$, then demonstration τ_i obtained a total reward greater than or equal to τ_j. We also sort $\mathcal{F}$ so that after sorting, $\mathcal{F} = \{\tau_1, \tau_2, ..., \tau_M\}$, and if $i \geq j$, then demonstration τ_i received a reward *less than or equal to* τ_j. The question to answer now, is how we can use these sorted sets of demonstrations to change how we compute the empirical feature counts? We do so by making an assumption, that the quality of the human demonstration follows a uniform distribution. Keeping this assumption in mind, recall Eq. (3),

$$\tilde{\mu}_k^{\mathcal{D}} = \frac{1}{N}\sum_{\tau\in\mathcal{D}}\sum_{t=1}^{h}\phi_k((s_t^\tau, a_t^\tau)) = \sum_{\tau\in\mathcal{D}}\frac{1}{N}\sum_{t=1}^{h}\phi_k((s_t^\tau, a_t^\tau)).$$

We can view $\frac{1}{N}$ as a weight on trajectory τ_i. We modify this equation to use a different weight θ_{τ_i} instead of $\frac{1}{N}$. Since we are assuming that the quality of demonstrations follows a normal distribution, let us consider n independent samples of a random variable $X \sim U(0, 1)$, and their order statistics. The expected value of $X_{(i)}$ is

$$\mathbb{E}[X_{(i)}] = \frac{i}{n+1}.$$

If we consider the sum:

$$\mathbb{E}[X_{(1)}] + ... + \mathbb{E}[X_{(n)}] = \frac{1 + ... + n}{n+1} = \frac{n(n+1)}{2(n+1)} = \frac{n}{2}.$$

Given this, we define our weight θ_{τ_i} on the ith trajectory τ_i in $\mathcal{D}$ and $\mathcal{F}$ as $\frac{\mathbb{E}[X_{(i)}]}{\frac{N}{2}} = \frac{2i}{N(N+1)}$ and $\frac{\mathbb{E}[X_{(i)}]}{\frac{M}{2}} = \frac{2i}{M(M+1)}$, respectively. Thus, in Inverse Reinforcement Learning through Ranking (IRLR), we use a modified computation of an empirical feature count:

$$\tilde{\mu}_k^{\mathcal{D}} = \sum_{\tau\in\mathcal{D}}\theta_\tau\sum_{t=1}^{h}\phi_k((s_t^\tau, a_t^\tau)). \tag{7}$$

In Inverse Reinforcement Learning through Failure and Ranking (IRLFR), we use Eq. (7) to compute the empirical feature expectations of the successful demonstrations, and use the following equation to compute the empirical feature expectations of the failed demonstrations:

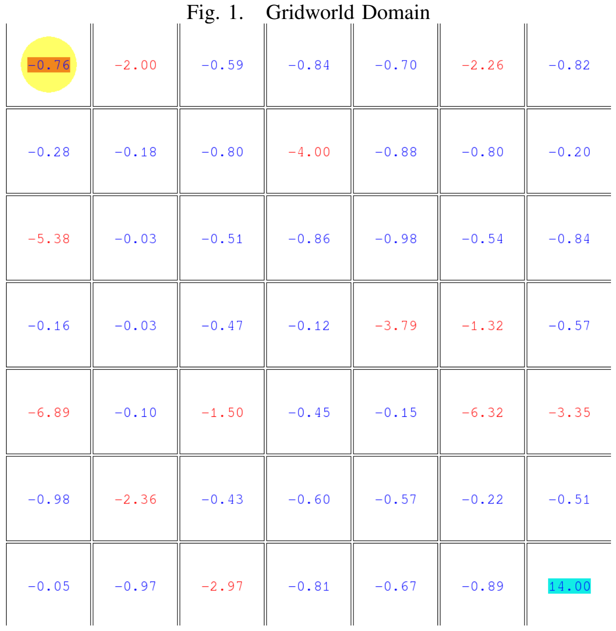
$$\tilde{\mu}_k^{\mathcal{F}} = \sum_{\tau\in\mathcal{F}}\theta_\tau\sum_{t=1}^{h}\phi_k((s_t^\tau, a_t^\tau)). \tag{8}$$

To summarize, IRLR is exactly MEIRL, except using Eq. (7) to compute the empirical feature expectations. IRLFR is exactly IRLFR, except using Equations (7) and (8) to compute the empirical feature expectations. Intuitively, the idea behind our computation is that the sum of the expected values of the $X_{(i)}$ can thought of as summing the rewards of the trajectories. The division by $\frac{n}{2}$ can be thought of as normalizing. Thus, the weight on each trajectory can be thought of as its portion of the total reward.

IV. Experimental Setup

A. Domain

Fig. 1. Gridworld Domain



To evaluate $IRLFR$, we implemented a 7x7 gridworld domain to run experiments. The agent starts in the top left square, and the goal state is the bottom right square of the screen. The agent's episode terminates upon reaching the goal state. The gridworld rewards are structured as follows:

- The terminal state has a reward of $rows + columns$. In our case, a reward of $7 + 7 = 14$.
- A nonterminal state s has a reward generated in the following way:
    - With probability 0.3, s is assigned a reward uniformly randomly drawn from the interval $[-7, -1]$. States with these rewards are highlighted red in the gridworld.
    - With probability 0.7, s is assigned a reward uniformly randomly drawn from the interval $[-1, 0]$. States with these rewards are highlighted blue in the gridworld.

Note that while the gridworld graphically depicts the reward, the reward is only available for the human demonstrator's benefit, and the learning agent is given no knowledge of the reward function.

The task is to complete an episode while maximizing the reward received in the episode. A *successful demonstration* is a demonstration that has a total reward that is nonnegative. All other demonstrations are classified as failures.

The features ϕ_k are defined as follows: $\phi_k(s) = 1$ if $s = k$, and $\phi_k(s) = 0$ otherwise, where the states are numbered $1 - 49$, row-wise. For example state 2 (first row, second column) would have feature vector $\phi(s) = (0, 1, 0, ..., 0)$, where $\phi_2(s) = 1$.

V. Experiments

For our experiment, we compare 5 different things against each other: the human demonstrator, MEIRL, IRLR, IRLF, and IRLFR. In each of ten different scenarios (10 different randomly generated gridworlds), a human (one of the authors) provides 6 demonstrations, 3 of which are successes, and 3 of which are failures. For MEIRL and IRLR, we only provide as input to the algorithms the successful demonstrations. For IRLF and IRFLFR, we provide all the demonstrations as input. For the algorithms involving ranking, we presort the demonstrations based on their actual score, and pass the ranked demonstrations (without the scores) to the algorithms. After learning a reward function and a policy, we have each of the agents perform the task for 50 episodes, and we take the average (actual) reward obtained by each of the agents, and compare them. We do not make quantitative comparisons regarding how much better each method performs relative to one another, because this is highly dependent on the reward function we define for the gridworld, which can differ largely between experiments. We focus on the simple metric of whether one method performs better than another. Our results are as follows:

TABLE I
My caption

Human Demonstrations	In all 10 experiments, the human's average reward is outperformed by all other agents.
MEIRL	In all 10 experiments, MEIRL performed worse than the other IRL agents. It only surpassed IRLFR two times, and was always surpassed by the other IRL methods.
IRLR	IRLR performs quite well, scoring the highest in 4 of the experiments, scoring the second highest in 5 of the experiments, and performing the worst amongst the IRL agents once.
IRLF	IRLF performed well, but had variance. It performed the best in 2 experiments, 2nd best in 5 experiments, 3rd best in 1 experiment, and 4th best in 2 experiments.
IRLFR	IRLFR also has variance, and it is more more pronounced. IRLFR performed the best in 4 experiments, 3rd best in in 4 experiments, and 4th best in 2 experiments.

Humans perform rather poorly, because the human's average performance includes both the successful and failed demonstrations. We can see that for the most part, IRLR, IRLF, and IRLFR all perform better than standard MEIRL, indicating that at the very least, these methods improve upon the baseline, on average. Interestingly, we find that methods that include learning from failure have higher variance in performance, perhaps due to the difficult-to-tune λ parameter. Additionally, many times IRLR and IRLF perform better than IRLFR, which seems to combine the best of both worlds. One possible reason for this is because failed demonstrations may not be as uniformly distributed as the successful demonstrations, which can cause the algorithm to compute potentially incorrect feature counts on the failed demonstrations when we assume its uniformity.

VI. Discussion and Future Work

In this paper, we introduced two new algorithms IRLR and IRLFR, the first of which weights the input demonstrations differently when computing the empirical feature counts, based on the rankings of the trajectories. IRLFR combines IRLR with IRLF, to create a combined framework that employs failed demonstrations and ranked demonstrations. We test our framework in a simple gridworld domain and show that IRLR and IRLFR perform better than MEIRL on average, and that modifying feature counts can potentially lead to improvements in the policy learning, but if the assumptions do not hold, can negatively affect policy learning.

In the future we hope to extend our methods to more complex domains, with larger state spaces, as opposed to the simple gridworld domain. Additionally, both the algorithms we introduced were made within the Maximum Entropy framework, a model-based framework. Future work will extend these to a model-free setting.

Additionally, while we are mitigating the assumption that the human is making optimal demonstrations, we are introducing the assumption that humans can make an accurate assessment of the trajectory's quality. Additionally, we assume that the quality of a human demonstration follows a uniform distribution. The distribution of the quality of demonstrations is highly task-dependent, and demonstrator-dependent, and potential future work can perform user studies on a variety of tasks to find a good model for the performance of demonstration. Additionally, there is literature on preference learning, and future work should seek to utilize these more sophisticated methods. Additionally, humans can make mistakes when ranking demonstrations, and we would like to extend our framework to have guarantees and still perform well *even when there are mistakes in the ranking of the input*. Furthermore, if we were to incorporate additional information, such as *how much* better one demonstrated trajectory is relative to another, perhaps we can achieve more insights. Additionally, future work should consider incorporated rankings between segments of demonstrations, and suggested in [6]. This can provide far more information for the learning agent to learn from.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] R. Akrour, M. Schoenauer, and M. Sebag. Preference-based policy learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 12–27. Springer, 2011.

[3] R. Akrour, M. Schoenauer, and M. Sebag. April: Active preference learning-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 116–131. Springer, 2012.

[4] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[5] M. Bogdanovic, D. Markovikj, M. Denil, and N. de Freitas. Deep apprenticeship learning for playing video games, 2015.

[6] M. Cakmak and A. L. Thomaz. Designing robot learners that ask good questions. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

[7] L. El Asri, R. Lemonnier, R. Laroche, O. Pietquin, and H. Khouzaimi. Nastia: Negotiating appointment setting interface. In *LREC*, pages 266–271, 2014.

[8] L. El Asri, B. Piot, M. Geist, R. Laroche, and O. Pietquin. Score-based inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 457–465. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[9] D. H. Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3804–3809. IEEE, 2011.

[10] D. H. Grollman and A. G. Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4(4):331–342, 2012.

[11] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in neural information processing systems*, pages 575–583, 2013.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[13] P. Odom and S. Natarajan. Active advice seeking for inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 512–520. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[14] A. Rai, G. De Chambrier, and A. Billard. Learning from failed demonstrations in unreliable systems. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 410–416. IEEE, 2013.

[15] K. Shiarlis, J. Messias, and S. Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1060–1068. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[16] R. van der Wijden. Preference-driven demonstration ranking for inverse reinforcement learning. 2016.

[17] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438, 2008.