University of Colorado at Boulder

Department of Computer Science

"Preference in IRL" - What Does It Mean?

Xinyu Cao

Supervisor: Bradley Hayes

A report submitted in partial fulfilment of the requirements
of the University of Colorado at Boulder for the degree of
Master of Engineering in Computer Science

November 21, 2021

Declaration

I, Xinyu Cao, of the Department of Computer Science, University of Colorado at Boulder, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of CU Boulder and public with interest in teaching, learning and research.

Xinyu Cao
November 21, 2021

Acknowledgements

Through out the course of this thesis, I received a great amount of help from many parties around me.

I would like to first express my gratitude to my advisor, Dr. Bradley Hayes, for providing me with guidance and support within the past two years. Without you, it would be impossible for me to deliver this work by myself. You are not only a great advisor, but also a wonderful friend. Thank you for helping me through the hardest time in my life.

I would also like to thank all my colleagues in the Collaborative AI and Robotics Lab. Thank you all for always being there whenever I want a discussion, and provide constructive suggestions and feedback on my work.

Finally, I would like to thank my mother and my friend Yanhao Jiang for supporting me and trusting my decisions.

Abstract

Preference learning has become a very popular topic when coming into solving a real world problem. State-of-the-art reinforcement learning algorithms are able to find the optimal solution given an environment and a target state, however, when human coordinators are involved in the task, a question that the agent must be able to take into consideration is: Which solution does the human participant prefer? This is a problem of great significance, as many real world problems involve both the human and the agent to solve, and understanding human's preference towards the current task is critical to success.

Many studies these days attempt to find a solution to this problem, however, the true definition of "preference" still remains ambiguous. Almost all of the works out there lack a clear explanation of what "preference" actually means in a real world context. At the high level, our study aims to define human preference as the cause of the difference in understanding of the environment between human and the agent. In other words, preference can be viewed as an underlying factor that makes the human arrive at a diverse reward function than the robot agent. We also prove that, under some environmental settings, preference cannot exist, thus cannot be learned.

On the other hand, we found that preference is never being evaluated in any existing work. Human beings are not exactly the expert in making the best decision, and they, in many possible ways, could have unreasonable preferences. In this work, we recognize this problem and propose a method in which the goal is to be able to find an equivalent of the human's preference of a specific task on the agent's reward function, which serves the purpose of evaluating whether that preference is compatible with the environment, and whether if it is safe to be applied. We started with

defining a threshold reward value that represents the basic completion of the task. Then, given a fully observed and controlled system and a target state, we learn an optimal policy through reinforcement learning, which has the highest expected rewards among all policies. These values will be used later on as the metrics to evaluate the reward function learn from a set of expert demonstrations through inverse reinforcement learning, in which the human demonstration can be suboptimal with respect to the current environment. In the end, our method should succeed in classifying the underlying human preference as compatible or incompatible with the environment.

We also propose a simple task environment in which the concept explained above could be tested in, but the actual implementation is yet unavailable at the time of this defense. However, we believe that our proposed definition and the evaluation method will still hold true in other more complicated real task environments.

Keywords: Preference Definition, Preference Evaluation, Preference-based Learning, Inverse Reinforcement Learning, Suboptimal Demonstrations Learning

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Human preference plays an important role when coming into solving a collaboration task in the real world. Given a fully observed, single-agent task environment, a state-of-the-art robot is able to learn the optimal policy using a reinforcement learning algorithm. However, when a human collaborator is present, it will be critical for the robot agent to be able to recognize the impact of the human's preference on its policy, and update its belief of the environment to be closer to the human's belief when needed. This will improve the efficiency of completing the task, as the robot is able to provide help in accordance with the human's expectation.

Many outstanding researchers recognized this problem, and aimed to solve it with different methods, thus preference-based learning has become a very popular field of study(5)(10)(14)(18)(7)(12)(16)(19). While some algorithms use active querying to train a model based on human responses (5)(10)(14)(18), others use pairwise trajectory comparison which assign preference labels to demonstrated trajectories, or assign labels toward each state in a trajectory(7)(10)(12)(14)(18)(16). However, these methods start with an assumption that the human's preference is always adaptable. Such methods have some shortcomings: 1. trajectories, demonstrations and preference labels are very hard to obtain when having a very complicated task domain, and 2. these methods requires the human participants to have a certain level of expertise in the task, in order to provide reliable feed-

back and labels. On the other hand, without preference ever being evaluated, these methods still able to achieve some level of success because they act ahead to remove unsuccessful trajectories to prevent poor preferences, which are the trajectories that have very low cumulative reward, resulting in very undesirable consequences in reality if performed that way. By eliminating these trajectories, these methods are able to handle poor preferences since they do not offer the corresponding choice to the human. If these trajectories are not eliminated, then evaluating preference becomes very important. In our work, we propose a method that is able to evaluate whether the human's preference is actually compatible with the task environment.

While some preference-based learning algorithms focus on learning only the policy, others work on learning a reward function that can best describe human behavior, as a result, inverse reinforcement learning plays an important role in these methods(1)(2)(3)(4)(8)(11)(13)(15)(20)(21)(22). Inverse reinforcement learning takes demonstrations, and works backwards to learn a reward function. However, IRL has a significant shortcoming, that is, it always assumes the demonstrations to be optimal when learning the reward function. As a result, if given suboptimal demonstrations, the learned reward function will also be impacted. Currently there are some works that target the problem of suboptimal IRL(8)(15)(21), we mention suboptimal learning here because, if a participant demonstrates a task with underlying preferences that have impacts on his behaviors, then it is very likely that the resulting demonstrations are suboptimal with respect to the task environment. This problem is rarely mentioned in any preference-based learning methods that either utilize IRL to learn or simply directly approximate reward functions. We state that, in the process of preference learning, humans are always acting optimally in their understanding when providing demonstrations, but the resulting demonstrations may or may not be optimal with respect to the original task environment. We will discuss into details about the situation where the approximated reward function is suboptimal, and prove that, given each scenario, our method is able to evaluate the learned reward functions, and arrive at a conclusion of whether the reward function should be adapted by the agent.

Upon introducing preference-based learning and IRL, we realized pref-

erence is never being thoroughly defined anywhere. In the context of real world robotics problems, we state that there does not exist such a thing as preference. Instead, we define preference in a new way that is more closely related to real world robotics problems. That is, preference is interpreted as the difference in the belief of the environment between the agent and the human participant. We conclude that, preference can only be interpreted as a tiebreaker, which leads to non-identical reward functions between the human and the agent. On top of this, we expand this definition further into details. We realize that, some work focuses on getting the agent and human participant to arrive at the same level of comprehension of the environment(17), by tutoring the human to reach the same level of understanding of the environment as the agent. This method start with an assumption that the human's policy deviates from that of the agent, due to the reason that the human does not have the same amount of knowledge of the task environment as the agent. This assumption reminds us of another important factor to consider when talking about human preferences, that is, under the different scenarios: 1). humans have the full understanding of the true reward function R, 2). Humans only have partial understanding of the true reward function R, and 3). true reward function R is completely unknown by the human, what role does preference play in each scenario? Can preferences still exist for each case? We not only propose a new definition of preference, but also expand it into more details that will be able to answer these questions with respect to each scenario.

1.2 Scope

We divide the scope of this project into five parts.

- Propose a new and thorough definition of preference.
- Model preference as a reward function $\hat{R}$ learned from IRL.
- Design an evaluation algorithm and threshold values.
- Case by case evaluations.
- Propose a sample test environment

In this project, our goal is to provide a definition of preference as well as a new way of learning and evaluating preference. Instead of designing an algorithm or model that focuses on learning solely the human's preference, we focus on providing a distinct and thorough definition of preference before we dive into actually learning the preference itself. On top of that, we also introduce a pipeline that is able to learn and evaluate human preferences on a given task. FIGURE

1.3 Overview

This thesis paper is organized as follows. In chapter 2, we divide all related literature into a set of topics: Preference-based learning, Inverse reinforcement learning, and Suboptimal demonstrations learning. We will discuss their relationship with our project respectively. In chapter 3, we will dive into demonstrating and explaining the details of our proposed definition and evaluation methodology, including case by case evaluations. Chapter 4 will talk about the simple task environment we propose to build as a sample environment to test our methodology, and will provide general outlines as well as explaining the details of the algorithms that we suggest using. Finally, Chapter 5 will illustrate all the future works.

Chapter 2

Related Work

2.1 Preference-based Learning

Preference-based learning is a sub field of learning where the goal is to focus on learning the preference information. There is a great amount of work focusing on learning the preferences using active querying(5)(10)(14)(18), or through rankings (7)(10)(12)(14)(18)(16). Other preference-based reinforcement learning methods are studied and evaluated by Wirth et al.(19). These methods focus on learning a reward function that best infer the human's behavior. Some of them did so by constantly querying the human throughout the task execution for feedback about the current trajectory (5)(14)(18), or even more straightforward, asking the human about which action they prefer at a given state(10). Others did so through labeling trajectories and rankings, where the users are asked to rank a given set of trajectories(7), or simply do a pairwise comparison(5)(12)(16)(18), and the results are used to learn or update the reward functions.

However, these methods are usually hard to realize since 1). Informative queries are hard to design 2). Labeling trajectories requires too much time, especially for more complicated tasks that have a very large state space and set of possible trajectories 3). These methods require the human participant to have a high level of expertise, such that they are able to provide reliable feedback and labels. These algorithms also requires experiment designers to be proactive and eliminate suboptimal trajectories to prevent poor preferences.

2.2 Inverse Reinforcement Learning

Inverse reinforcement learning is the field of learning, where the goal is to learn a reward function given a set of policies, or expert demonstrations (1)(2)(3)(4)(8)(11)(13)(15)(20)(21)(22). The state-of-the-art IRL algorithm is able to resolve the ambiguity problem using maximum entropy, where the expert demonstrations can be described with multiple different reward functions(20)(22), we suggest using these methods for testing our concepts. Existing works have combined inverse reinforcement learning with preference learning(12)(16), where demonstrations are used to learn a reward function through IRL, and the feedback from queries or ranked trajectories are used to update the estimated reward functions.

Notably, IRL faces some challenges, as great demonstrations are generally hard to obtain from users, and since IRL always assumes the expert demonstrations to be optimal when learning a reward function, it is not very robust when the demonstrations are suboptimal. When IRL is used to learn preference, suboptimal demonstrations are usually the case in which we will be facing. Our work is related to IRL in that expert demonstrations encodes preference information as they describe what the human wants to do, and learning a reward function from these demonstrations should allow the agent to act in accordance to the person's preference. In other words, we state that the learned reward function, $\hat{R}$, will encode human's preference towards performing a certain task.

2.3 Suboptimal Demonstrations Learning

As a sub-field problem of IRL, learning from suboptimal demonstrations focuses on solving the problem where if the demonstrations are suboptimal, the resulting reward function will also be suboptimal. This is not a very popular field of study. Existing approaches include learning from the failed demonstration(8)(15), where they treat the failed demonstrations as a negative bias. Instead of maximizing the similarity between the learned policies to those of demonstrated, they work towards the opposite direction to avoid the failed demonstrations while matching the successful ones. Other methods take into consideration the behavior noise when performing

IRL(21), where the reward function is estimated from the demonstrations, but noisy demonstrations are marked with low reliability and ignored during reward learning.

However, when it comes to learning preferences, ignoring suboptimal or failed demonstrations is not exactly the desired way, as they still encode preference information and should still be considered and evaluated. The present work differs from these methods in that, we consider expert demonstrations to encode preference information, and perform IRL to learn a reward function regardless of whether these demonstrations are optimal, suboptimal, or failed. Because of the nature of IRL, we do not directly utilize the learned reward function, instead, we propose taking an extra step to evaluate the reward function, where we design an algorithm to infer the tolerance of the underlying preferences.

2.4 Reward Function Disparity

Some works dedicate their effort in identifying the differences between reward functions(6)(9), or, exploring algorithms in bringing human participants to agree with the agent when non-identical reward functions exist through explainable AI(17). These algorithms quantify the difference between reward functions, in which reward functions are evaluated via the distances from other reward functions(6), or visualize the differences through plotting(9). In our work, we define preferences as the proof that non-identical reward functions exist. More specifically, we state that, preference acts as a tiebreaker in human's decision making process, resulting in a reward function $\hat{R}$ that is different from the true reward function R. Unlike other works, we do not focus on quantifying and measuring the difference between reward functions, or focusing on bring human's belief towards the ground truth. Rather, we focus on evaluating the reward function learned from IRL, before allowing the agents to update their belief towards that of the human's.

Chapter 3

Methodology

Summary

Our work can be divided into two parts: 1. proposing a new and thorough definition of preference 2. designing an reward function evaluation algorithm. In this section, we will first dive into introducing and explaining the definition of preference, and discuss its nature with respect to three scenarios: R fully observable, R partially observable, and R unknown. We will also show the cases where suboptimal reward functions may take place, and this will lead to our next topic, where we focus on explaining the details of the reward function evaluation algorithm.

3.1 Preference Definition

Consider a real world scenario, where researchers are setting up an environment and trying to hold user studies, by recruiting participant to perform a designed task in the environment. In this case, there are only certain information that are already known to the researchers: 1). the environment setup, including the true reward function R, 2). the demonstrations collected your participant, and 3). the reward function $\hat{R}$ learned from the demonstrations using IRL. What remains unknown, is the process of the participant arrives at this set of demonstrations, or, the his decision making process that result in the set of demonstrations collected. When coming into preference learning, this process is the most critical part, as it

is where preference actually takes place and affects the human's behavior. Thus, we focus on modeling this underlying decision making process, and then define preference on top of it.

3.1.1 Definition

In order to better illustrate our definition of preference, consider this real world example, where you would like to purchase a cup of coffee, and there are two choices: decaffeinated or caffeinated. If both decaffeinated and caffeinated are \$5, which one would you choose? What about the case where decaffeinated is \$4 and caffeinated is \$5? In this case, how would you make the decision?

We start by modeling the decision making process, in order to do that, we must introduce a set of terms and their definitions, as illustrated below:

- **R**: R denotes the true reward function. For all IRL problems, there always exists a true reward function R
- $\hat{\mathbf{R}}$: $\hat{R}$ denotes to the reward function that models the preference, it is learned through IRL from the set of expert demonstrations.
- $\bar{\mathbf{R}}$: $\bar{R}$ is a self-designed, conceptual reward function that is derived from the true reward function R. Representing the intermediate state of the human's decision making process.
- **Tie**: we define tie as the equivalence between reward values. Given the example above, a tie will exist in the true reward function R, when both decaffeinated and caffeinated has the same price. A tie will be created in $\bar{R}$, when decaffeinated and caffeinated have different price, but the difference is neglected by the human, generating a conceptual tie in $\bar{R}$.

With all the important terms explained, we are able to introduce our definition of preference. Mainly, to define what preference is in an IRL problem, and under what situation can it exist and be learned. We state that:

1. **If at any time, human agrees with the true reward function R, then preference does not exist.**

2. **If preference can be defined, then either:**

   (a) **The true reward function R contains ties.**

   (b) **Human disagree with the true reward function R, and the intermediate reward function $\bar{R}$ is different from R AND contains ties.**

3. **Preference can only be defined as a *tie-breaker*, that breaks the ties between policies, resulting in a reward function $\hat{R}$, where $\bar{R} \xRightarrow[\textbf{Preference}]{} \hat{R} \neq R$**

Note that, $\bar{R}$ contains ties, that are either inherited from the true reward function R, or logically created by the human due to disagreement with R. Preference is defined as the tie-breaker, which breaks the ties in $\bar{R}$, arriving at the final reward function $\hat{R}$, which is the reward function that is being optimized by the human participant when performing the task, and also the reward function we learned from the set of demonstrations, under the assumption that IRL algorithm is at its best functionality. It is also worth noting, where in any cases, if $\bar{R} = \hat{R}$, we say that preference cannot be defined. Since based on the definition provided above, if $\bar{R} = \hat{R}$, then either:

- $\bar{R}$ contains no ties at all.
- Preference is not used to break the ties.

In order the simplify the problem, we assume that preference will always be used to break ties if they exist. The first situation is just another way of saying human is agreeing with the true reward function R, corresponding to definition [1].

Given the definitions of preference, we are also interested in apply them into different settings that are frequently seen in real world experimental designs, which are: R fully observable, R unknown, and R partially observable. We will discuss them respectively in next three sections.

3.1.2 R Fully Observable

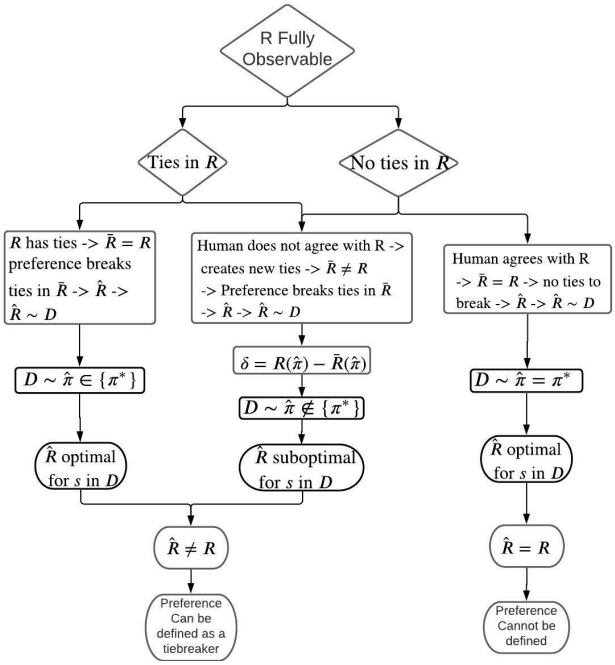

Figure 3.1: Preference Under Fully Observable R

We first would like to talk about the "R Fully Observable" scenario, where in this setting, the true reward function R is given to human participants prior to demonstrating. In other words, human participants and the agent start at the same level, where they have the same amount of knowledge of the task environment.

For this scenario, we are able to visualize the nature of preference in a very clear way, as shown in Figure 3.1. The leftmost branch of the flowchart corresponds to definition [2a], where the true reward function R contains ties, $\bar{R}$ inherits the ties from R, and preference breaks the ties, arriving at the final reward function $\hat{R}$. In this case, $\hat{R}$ will always be optimal. Consider the coffee example, when decaffeinated and caffeinated coffee have the same price, there exist multiple policies that can optimize R. In this case, getting decaffeinated or caffeinated will result in the same amount of cumulative reward. Therefore, no matter which policy that preference guides the human towards, the resulting estimated reward function $\hat{R}$ will always be optimal.

The middle branch corresponds to definition [2b], where new ties are

created by the human due to disagreement with R. Consider when decaffeinated and caffeinated coffee has different prices, and the human disagrees with R because he thinks that the \$1 difference is neglectable, creating a new tie between decaffeinated and caffeinated in $\bar{R}$. It is very likely, in this case, that the resulting estimation of $hatR$ is suboptimal. Because it is possible that the human prefer caffeinated over decaffeinated, while the true reward function R can only be optimized if decaffeinated is selected. This is the scenario where an evaluation on the estimated $\hat{R}$ is required.

Finally, the rightmost branch corresponds to definition [1], where preference cannot be defined because the human agrees with R. In this case, participant's behavior is entirely encoded in R, and does not reflect any preference at all.

3.1.3 R Unknown

In the setting where R is completely unobservable to the participants, preference cannot be defined, and we will not be able to learn preference through the estimated reward function $\hat{R}$. This is because:

- Whether the true reward function R contains ties or not remains unknown.
- Disagreement against unknown R is not reasonable.

With R fully unobservable, the human participant is acting upon speculated reward values for all states. For example, the participant might guessed that the difference in reward value between decaffeinated and caffeinated coffee is 1, and creates a tie based on this conjecture. However, the true reward function R might be suggesting that the difference between decaffeinated and caffeinated is actually 5. In this case, the participant might not be able to neglect this difference any more, indicating the tie that was created based on the conjecture is a false tie, and we cannot say that the resulting $\hat{R}$ is encoding the correct preference.

3.1.4 R Partially Observable

Unlike R fully observable and R unknown settings discusses above, where we are able to draw clear boundaries between all possible situations, and

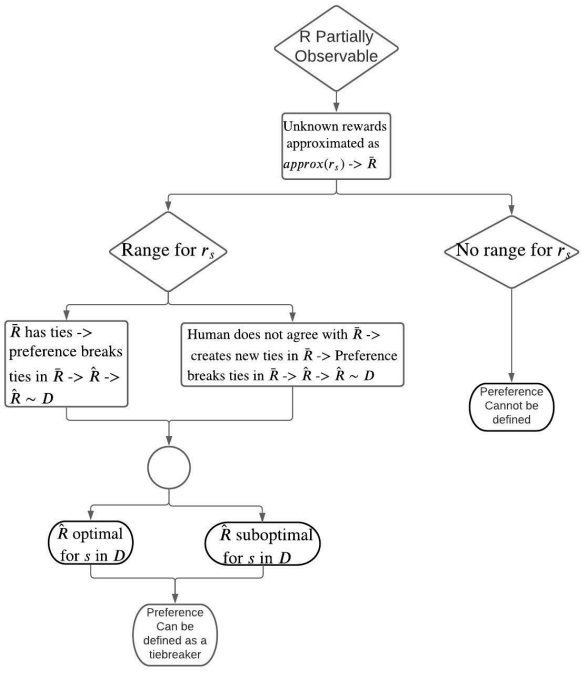

Figure 3.2: Preference Under Partially Observable R

illustrate the nature of preference respectively. Under the R partially observable setting, there exists many possible situation that each comes with great uncertainties. However, we are still able to categorize them roughly into two parts, as shown in Figure 3.2.

We state that, given partially observable R, the participants will act upon the conceptual reward function $\bar{R}$, in which the reward values of the unknown states are conjectured. The rightmost branch suggests that, if there exists absolutely no information about what the true reward value of an unknown state might be, then it can be treat the same as if R is unknown, where we say that preference cannot be defined at all, since the approximation of the reward values of the unknown states could deviate from the ground truth tremendously, causing false ties to be created.

In the contrary, the left branches represents the scenario, where the range in which the true reward value lies in is provided for the unknown states. In this case, the participant is able to make more reliable approximations about the unknown states, reducing the possibility of creating false ties. To better illustrate, consider the example, where the true reward value of getting the decaffeinated coffee is known as 4, and the participant

only knows that the true reward value of getting the caffeinated coffee lies within the range 5 ± 2. The participant is able to base his approximation on this range, and create more reliable ties. In this case, we say that the preference can exist, and be encoded in $\hat{R}$, but more study is required to dive into nuance cases in order to determine whether $\hat{R}$ is optimal or suboptimal. Thus, we conclude that, if at any time, preference can exist under R partially observable setting, evaluation is always required on the estimated reward function $\hat{R}$.

3.2 Reward Function Evaluation

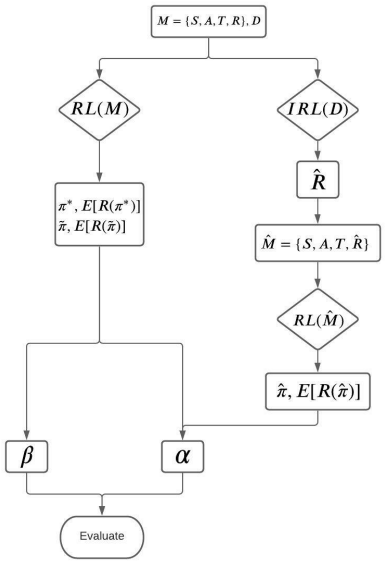

Figure 3.3: Evaluation Pipeline

Our evaluation method can be divided into three parts. As shown in Figure 3.3, we first focus on learning an optimal policy given the task environment, using a reinforcement learning algorithm. Meanwhile, inverse reinforcement learning algorithm is being applied on a set of expert demonstrations, and learns a reward function $\hat{R}$ describing user's behavior, and is expected to encode preference information. The last part involves calculating a threshold value α, which is used along with the threshold value β, to evaluate the learned reward function. Terms that appeared in this graph is explained below:

- RL: Reinforcement Learning
- IRL: Inverse Reinforcement Learning
- $M = S, A, T, R$: Markov Decision Process, with S = state space, A = action space, T = transition matrix, and R = true reward function.
- D: Set of expert demonstrations.
- π^*: Optimal policy given M.
- $\tilde{\pi}$: Least ideal policy given M. Policy that accomplishes the task with the lowest expected cumulative reward.
- $\hat{\pi}$: Optimal policy given $\hat{M}$.
- α, β: Evaluation thresholds.

3.2.1 Reinforcement Learning

Reinforcement learning is the method which learns an optimal policy that maximizes the cumulative reward of a task environment, a simple model is shown in Figure 3.4. Let $S, A, T, R \in M$ denotes state space, actions, transition probabilities, reward, and the markov decision process(MDP). Reinforcement learning aims to solve M for an optimal solution $\pi^* = [(s1, a1), (s2, a2), ...]$, which has the highest expected cumulative reward $E[R(\pi^*)]$.

Common reinforcement learning algorithms include Value iteration/Q-learning, Policy iteration, etc. We will explain more details about the RL algorithm we suggest using to build the test environment in the next section.

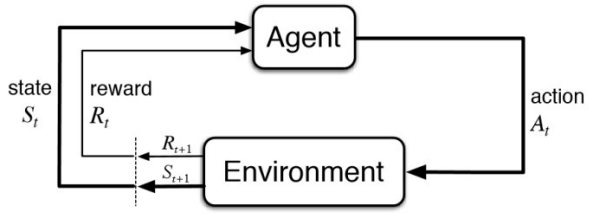

Figure 3.4: RL

3.2.2 Inverse Reinforcement Learning

To the contrary of traditional RL problems, which aims to learn policies from reward functions, the goal of inverse reinforcement learning is to learn a reward function from policies demonstrated by humans. For a traditional IRL problem, we assume policies demonstrated by humans are optimal, and the goal is to find a reward function that can best describe these policies, as shown in Figure 3.5. In our evaluation algorithm, IRL is used to estimate the reward function $\hat{R}$ from a set of expert demonstrations D, with the participant's preference acts as the tie-breaker throughout the demo process. Therefore, $\hat{R}$ is expected to encode the underlying preference information.

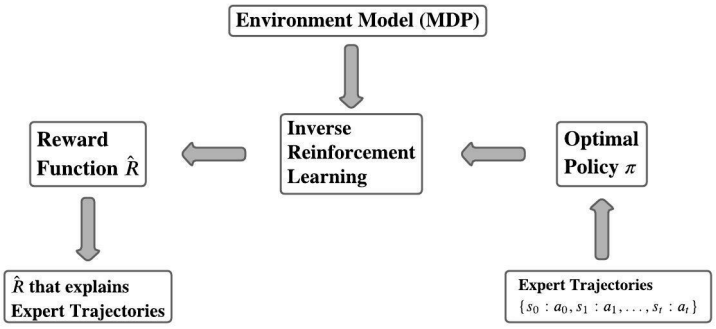

Figure 3.5: IRL

Current inverse reinforcement learning algorithms include apprenticeship learning(1), maximum margin planning(13), structured classification, etc. In many cases, using these methods, a single set of demonstrations can lead to multiple different reward functions that all are able to describe human behaviors. To solve this ambiguity problem, as introduced by Ziebart et al.(22), an approach which employs the principle of maximum entropy to choose a model whose distribution over paths is constrained only on the feature counts and nothing else. Maximum Entropy IRL has become the most widely used inverse reinforcement learning algorithm.

Built upon the Maximum Entropy IRL algorithm, Wulfmeier et al.(20) introduces a method which combines neural networks with Maximum Entropy to solve IRL problems. Reward function is approximated with a deep neural network architecture, with input as state features. The difference

in state visitation frequency between the policy learned using the current neural network and the user demonstrations is used as the loss to train the neural network. We will dive deeper into explaining the actual MaxEnt algorithm in the Chapter 4.

Deep MaxEnt IRL exceeds MaxEnt IRL in dealing with large-scale IRL applications where more complex reward functions are required, more specifically, Deep MaxEnt IRL can better handle nonlinear reward functions that are composed of a larger amount of features, since MaxEnt IRL relies on expressing reward function as a weighted linear combination of hand selected features. It also exceeds other IRL algorithms in computational speed, which also provides better computational performance when dealing with large state spaces and more complex reward function structures.

3.2.3 Baseline Threshold β

We introduce a brand new threshold measurement in our method, β, stands for the baseline value of the expected cumulative reward of the current task environment. In other words, it represents the baseline requirements on the current task environment in which a policy can be adapted.

Given the definition of the terms provided above, optimal expected cumulative reward value can be represented as $E[R(\pi^*)]$, and the expected cumulative reward value of the least ideal policy $\tilde{\pi}$ is represented as $E[R(\tilde{\pi})]$. We define β as the ratio of these two expected cumulative reward values:

$$\beta = \frac{E[R(\tilde{\pi})]}{E[R(\pi^*)]} \tag{3.1}$$

This threshold value represents the satisfaction of the baseline requirement of a particular task given M, it is one of the threshold measurements used to evaluate the learned reward function $\hat{R}$, as $\hat{R}$ must not only encode the preferences, but also at least still satisfies the basic requirements on top of that. β can also be a self-designed value, indicating a particular requirement, or additional constraints.

3.2.4 Preference Threshold α

On top of the baseline threshold β, we design another brand new threshold value, α.

Based on the definitions of terms provided above, let the expected cumulative reward value of the optimal policy π^* represented by $E[R(\pi^*)]$. Meanwhile, given a set of expert demonstrations $D = \{\zeta_1, \zeta_2, \zeta_3, ...\}$ with each demonstration ζ composed of a set of state-action pairs, and also encodes human's preferences. Let $\hat{R}$ denote the reward function estimated from D using the Deep MaxEnt IRL algorithm. Given that $\hat{\pi}$ is the policy optimizing $\hat{R}$, which can be interpreted as the desired policy of the human participant, and has its expected cumulative reward value represented by $E[R(\pi^{human})]$. We define α as the ratio of this value and the optimal expected cumulative reward value:

$$\alpha = \frac{E[R(\hat{\pi})]}{E[R(\pi^*)]} \tag{3.2}$$

This threshold measurement can be interpreted as the preference-equivalent threshold value, which is used to find where the policy learned from the estimated reward function $\hat{R}$ lies on the scale based on the true reward function R. Along with the baseline threshold ratio β, we are able to design an integrated evaluation metric to determine whether the agent should update its belief towards $\hat{R}$ or stay the same.

3.3 Case by Case Evaluation

In order to better illustrate how this evaluation algorithm works, we would like to do a case by case evaluation on a sample task environment.

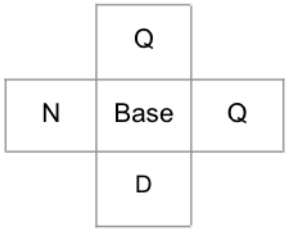

Figure 3.6: Evaluation Sample Environment

$r(s,a)$	State s	Action a	Next State $s^{'}$
Coin value in $s^{'}$	Base	Up/Down/Left/Right	Any $s^{'}$
0	Any s	Up/Down/Left/Right	Base
-1	Any s	Stay/INVALID	$s^{'} == s$
100	Any s	Up/Down/Left/Right	Terminal State

Table 3.1: Sample Environment R

Given: A gridworld environment modeled as a simple Markov Decision Process $M = \{S, A, T, R\}$, as shown in Figure 3.6. And a set of expert demonstrations D

- A: Action space {Up, Down, Left, Right, Stay}
- S: State space $\{s_1, s_2, ...\}$, where $s = [x, y, \#\text{quarter}, \#\text{dime}, \#\text{nickle}] \in S$
- R: True reward function, with reward values shown in Table 3.1
- Terminal State: $s = [\text{Base}_x, \text{Base}_y, \#\text{quarter}, \#\text{dime}, \#\text{nickle}]$ where $sum(\#\text{quarter}, \#\text{dime}, \#\text{nickle}) = 2$
- D: A set of policies $\{s_1 : a_1, s_2 : a_2, ..., s_t : a_t\}$
- Outputs: $\hat{R}, \hat{\pi}, \alpha, \beta$

From Figure 3.1 and Figure 3.2, we conclude that, evaluation on the estimated reward function $\hat{R}$ is required when it is suboptimal. In the next sections, we are going to mainly focus on evaluating suboptimal $\hat{R}$, since if $\hat{R}$ is optimal with respect to M, it is certainly going to be adaptable.

3.3.1 $\hat{R}$ Adaptable

Given the environment above, based on the evaluation pipeline, we are able to get the below information required by the evaluation algorithm:

- $\pi^* = \{\{s_0 : Up, s_1 : Down, s_2 : Right, s_3 : Left\}, \{s_0 : Right, s_1 : Left, s_2 : Up, s_3 : Down\}\}$
- $\tilde{\pi} = \{s_0 : Left, s_1 : Right, s_2 : Down, s_3 : Up\}$

- $E[R(\pi^*)] = 50 + 100 - 0 = 150$
- $E[R(\tilde{\pi})] = 15 + 100 - 0 = 115$

Assume that, using IRL, the estimated reward function $\hat{R}$ from D, suggests the optimal policy $\hat{\pi}$:

- $\hat{\pi} = \{s_0 : Down, s_1 : Up, s_2 : Right, s_3 : Left\}$
- $E[R(\hat{\pi})] = 35 + 100 - 0 = 135$

We are able to calculate α to be $\frac{E[R(\hat{\pi})]}{E[R(\pi^*)]} = 0.9$ based on Equation 3.2, and β to be $\frac{E[R(\tilde{\pi})]}{E[R(\pi^*)]} = 0.77$ based on Equation 3.1. We state that, since β represents the baseline requirement, and $\beta < \alpha < 1$. This indicates the policy $\hat{\pi}$ lies within the acceptable range. Thus the estimated reward function $\hat{R}$ is safe to be adapted.

Recall that we modeled the decision making process as the fundamental to define preference in Section 3.1, to further prove that our definition is correct, we would like to relate it with this example.

We start by identifying ties in the true reward function R. According to Figure 3.6, we can see that there exists a tie between selecting the "Up" and action and the "Right" action, since both actions have expected reward values to be 25. As mentioned in Section 3.1, the intermediate reward function $\bar{R}$ may contains ties that are either inherited from R, or newly created due to disagreement with R. By looking at policy $\hat{\pi}$, we can see that the participant chooses to go "Down", picking up the dime, before going to "Right" or "Up" for the quarter. Thus, we can say that, $\bar{R}$ in inherits the tie $r(s, Up) == r(s, Right)$ from R, and at the same time, the participant thinks that he can neglect the value difference between the dime and the quarter, creating a new tie $r(s, Down) == r(s, Up) == r(s, Right)$ in $\bar{R}$. Finally, preference comes in as the tie-breaker to break this tie, resulting in the final policy $\hat{\pi}$ that optimizes the reward function $\hat{R}$, in which dime has a higher reward value than quarters: $r(s, Down) > r(s, Right) > r(s, Up)$.

3.3.2 $\hat{R}$ Unadaptable

Given the same environment, the set of information we are able to obtain is the same as the previous section:

- $\pi^* = \{\{s_0 : Up, s_1 : Down, s_2 : Right, s_3 : Left\}, \{s_0 : Right, s_1 : Left, s_2 : Up, s_3 : Down\}\}$
- $\tilde{\pi} = \{s_0 : Left, s_1 : Right, s_2 : Down, s_3 : Up\}$
- $E[R(\pi^*)] = 50 + 100 - 0 = 150$
- $E[R(\tilde{\pi})] = 15 + 100 - 0 = 115$

What is different, is the optimal policy $\hat{\pi}$, that optimizes the estimated reward function $\hat{R}$, learned from D. In this case, assume $\hat{\pi}$ is:

- $\hat{\pi} = \{s_0 : Down, s_1 : Up, s_2 : Stay, s_3 : Stay\}$
- $E[R(\hat{\pi})] = 10 - 2 = 8$

We can see that, the policy suggested by the estimated reward function $\hat{R}$ only picks up one coin, resulting in $\alpha = \frac{E[R(\hat{\pi})]}{E[R(\pi^*)]} = 0.0.053$, based on Equation 3.2. We state that, since $\beta > \alpha$, indicating the policy $\hat{\pi}$ resides below the baseline, thus the estimated reward function $\hat{R}$ should not be adapted.

We again relates our preference definition with this example. Given the same environment $M3.6$, as mentioned in the previous section, R contains the tie $r(s, Up) == r(s, Right)$. By looking at policy $\hat{\pi}$, we can see that the participant chooses to go "Down", picking up the dime, and then stay until the end of the timestamp. In this case, $\bar{R}$ contains the tie $r(s, Up) == r(s, Right)$ inherited from R, and also a new tie created by the participant, $r(s, Down) == (100 + r(s, Right)) == (100 + r(s, Up))$. Preference comes in to break the tie, resulting in the final policy $\hat{\pi}$ that optimizes the reward function $\hat{R}$, in which dime has a higher reward value than that of the quarters and reaching the terminal state combined: $r(s, Down) > (100 + r(s, Right)) == (100 + r(s, Up))$. This certainly does not look reasonable, and is also suggested so by the evaluation algorithm, where the output of the evaluation algorithm says that $\hat{R}$ is not adaptable because α is below the baseline threshold β. This proves that our preference definition and evaluation algorithm both are correct.

Chapter 4

Experiment

Summary

In this section, we design a gridworld environment same as the one mentioned in Chapter 3. We would explain the sample environment that we suggest others to implement in order to test our method. In general, we will talk about environmental setup, dive deep into explaining the Q-learning algorithm and the Deep MaxEnt IRL algorithm which we recommend using.

4.1 Environmental Setup

We formulate our problem as an application of a simple Markov Decision Process(MDP) with deterministic rewards. We design a very simple gridworld environment to test our methodology, as shown in 3.6. The general problem formulation is:

- Input: $M = \{S, A, T, R\}, D$
- Output: $\alpha, \beta, \hat{R}, \pi^*, \hat{\pi}$

We setup our environment following the guidelines below:

- The agent must start at the base, and must return to base in order to move to another cell.

- If not given a specific number of steps to act, the default task will be to visit all cells once. Otherwise, if given n-steps to act, the task would be to pick up n coins total.

Due to the setup that allows the robot agent to jump to different cells, we must design our state vector to be able to remember all cells that it has already traveled to. In this case, our state vector is represented as $s = (x, y, coin, [count1, count2, \dots]) \in S$, where each entry represents:

- x - x coordinate
- y - y coordinate
- coin - coin type in the current cell
- $[count1, count2, ..]$ - an array of count for each type of coin that has been collected by visiting a cell, with the array length equal to the total number of unique coin types in the environment.

As a result of the unique design of the state vector, we have a very large state space. Therefore, we decide not to define the entire state space, and only populate our state space as we proceed with the task. Our action space, represented be a set of integers $\{0, 1, 2, 3, 4\}$, with each number denotes $\{Up, Down, Left, Right, Stay\}$. To reduce the complexity of the environment, we set the transition noise to be 0, and assign reward values based on the coin value at that cell, with details of R shown in Table 3.1.

For this simple example, we use only one expert demonstration T, composed of a sequence of cells that the human participant has gone to. Then generates the expert policy D, which is a set of state-action pairs, based on this demonstrated trajectory T.

4.2 Q-Learning

As discussed in Section 3.2.1, there exists many algorithm for solving reinforcement learning problems. Due to the fact that it is very difficult to define the entire state space given our special environmental setup, we do not suggest using the standard value iteration algorithm to learn the optimal policy. Q-learning does not require the complete state space to do the

computation, and allows us to build up the q-table with only the states that have been visited, thus best suits our environment.

Given at time $t \in T$, the current state is denoted as s_t and the current action as a_t. Q values for all states are initialized to be 0, and is updated with the Bellman equation at each time step:

$$Q(s_t, a_t) = (1 - \alpha) * Q(s_t, a_t) + \alpha * (r_t + \gamma * max_a Q(s_{t+1}, a)) \tag{4.1}$$

Where α denotes the learning rate, and γ denotes the discounted factor. This equation is similar to a simple value iteration update, with the new Q value calculated as the weighted average between the old Q value and the future discounted reward. Detailed Q-Learning algorithm is shown in Algorithm 1

Algorithm 1 Q-Learning

Input: $M = \{S, A, T, R\}$
Output: Q

```
Q ← initialized to all 0 for each actiona
E ← episode
while E ≠ 0 do
    s ← current state
    while s ≠ Terminal State do
        if random.uniform(0, 1) less than ϵ then
            a ← random(A)
        else
            a ← argmax(Q[s])
        end if
        s' ← T(s, a)
        Q(s, a) ← Bellman Equation 4.1
        s ← s'
    end while
end while
return Q
```

Note that the Q-learning algorithm is used to learn the optimal policy

π^*, as well as within the Deep MaxEnt IRL to calculate the loss, as shown in Algorithm 2 Line #4. The former uses the reward function R, and the latter uses the neural network to approximate the reward values. The resulted Q-table is used to find the deterministic optimal policy π^*, and the intermediate policy, π which is used to calculate the state visitation frequency and the loss for training the neural network.

Algorithm 2 Maximum Entropy Deep IRL

Input: $\mu_D^a, f, S, A, T, \gamma$

Output: optimal weights θ^*

1: $\theta^1 =$ initialise_weights()

Iterative model refinement

2: **for** $n = 1 \rightarrow N$ **do**

3: $r^n =$ nn_forward(f, θ^n)

Solution of MDP with current reward

4: $\pi^n =$ Q-Learning(r^n, S, A, T, γ)

5: $\mathbb{E}[\mu^n] =$ propagate_policy(π^n, S, A, T)

Determine Maximum Entropy loss and gradients

6: $\mathcal{L}_D^n = \log(\pi^n) \times \mu_D^a$

7: $\frac{\partial \mathcal{L}_D^n}{\partial r^n} = \mu_D - \mathbb{E}[\mu^n]$

Compute network gradients

8: $\frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n} =$ nn_backprop$(f, \theta^n, \frac{\partial \mathcal{L}_D^n}{\partial r^n})$

9: $\theta^{n+1} =$ update_weights$(\theta^n, \frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n})$

10: **end for**

4.3 Deep Maximum Entropy IRL

Based on the evaluation algorithm, we are estimating a reward function $\hat{R}$ through IRL from expert demonstrations D. We state that, $\hat{R}$ will encode the participant's preference for the given task, and the policy learned from it, $\hat{\pi}$, will be the policy that the human will perform given the underlying preference. We suggest using the Deep MaxEnt IRL algorithm, as it not only can resolve the ambiguity issue, but also is able to deal with more complex non-linear reward functions than the general MaxEnt IRL algo-

rithm. Algorithm 2 describes the method proposed by Wulfmeier et al.(20) in details. Note that r^n stands for reward values of all states, where each value is the reward of a single state, obtained by using that state as the input to forward propagate through the neural network.

4.3.1 Neural Network Structure

As proposed by Wulfmeier et al.(20), we suggest using a fully connected neural network, the general structure of the neural network can be designed as the one shown in Figure 4.1. The input to the neural network is the state vector, in our case, given three types of coins, input vector will have a dimension of 1×6. This neural network has an input layer, two hidden layers, and an output layer. However, the number of hidden layers and the number of neurons per layer are not fixed values, tuning on these values are required in order for the neural network to achieve the best performance. We also suggest applying the RELU activation function to the outputs of each layer other than the final output layer, adding non-linearity to the estimated reward function.

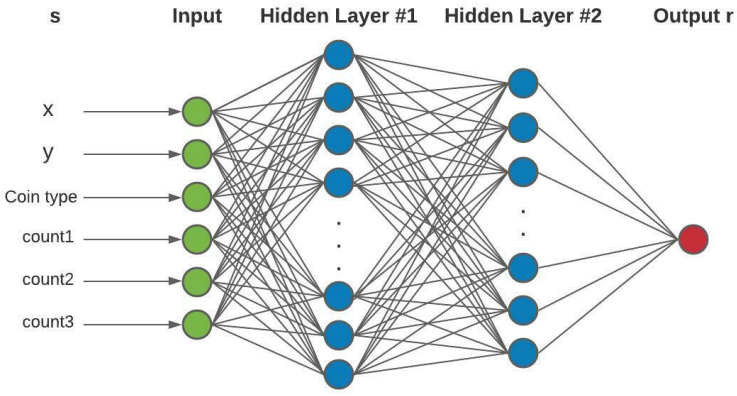

Figure 4.1: Fully Connected Neural Network

4.3.2 State Visitation Frequency

Line #5 in Algorithm 2 calculates the state visitation frequency of policy π, which is the optimal policy learned using Q-learning given the current neural network as the approximated reward function, as suggested by Wulfmeier et al.(20). Detailed policy propagation algorithm is illustrated

in Algorithm 3. The mean squared error between the expected state visitation frequency $\mathbb{E}[\mu^n]$ and the expert state visitation frequency μ_D can be used as the loss to train the neural network. Other loss functions such as mean absolute error can also be applied instead, but tuning is required to find the best loss function to use.

Algorithm 3 Policy Propagation

1: $\mathbb{E}_1[\mu(s_{start})] = 1$
2: **for** $n = 1 \rightarrow N$ **do**
3: $\quad \mathbb{E}_i[\mu(s_{goal})] = 0$
4: $\quad \mathbb{E}_{i+1}[\mu(s)] = \sum_{s',a} T(s, a, s') \ \pi(a|s') \ \mathbb{E}_i[\mu(s')]$
5: **end for**
6: $\mathbb{E}[\mu(s)] = \sum_i \mathbb{E}_i[\mu(s)]$

Chapter 5

Conclusions

In this work, we first introduce a definition for preference in the context of real world IRL problem, where we done through modeling the human's decision making process while providing demonstrations, and then identify the role of preference as a tie-breaker in this process. Upon introducing this new definition, which connects preference to reward functions, we also discussed the relationship between preference learning and suboptimal inverse reinforcement learning. We apply this definition into three settings that are frequently applied to real world experiments: R Fully Known, R Unknown, and R partially Known, and then identified all the cases where the estimated reward function $\hat{R}$ is optimal, emphasizing the importance of evaluating $\hat{R}$. From there, we introduce an algorithm that is able to evaluate $\hat{R}$, by mapping $\hat{\pi}$ onto the true reward function R. Our work is the first that provides a thorough definition of preference in IRL problems, and puts forward a preliminary method that helps researchers to deal with human preference in a more cautious and practical way.

Chapter 6

Future Work

6.1 R Partially Unknown

As mentioned in Section 3.1.4, we are unable to categorize R partially unknown into more precise cases, as there exists great uncertainties in $\hat{R}$ under this setting. In the future, we plan to explore more nuance cases, and strive to find edge cases where our definition of preference might not hold true.

6.2 Environmental Design and Real World Application

We proposed a sample environment in Chapter 4, for the actual implementation still remains to be done. In order to make preference shown in a more explicit way, we hand coded a feature to each cell. Current experimental design result in a state vector with very large dimensionality, and also result in a very large state space, which in turn increases the computational cost. What we would first like to accomplish in the future is to try implementing the sample environment using the design proposed in Chapter 4, and on top of that, adjust the setup to reduce the complexity in implementation.

On the other hand, we choose to design a very simple environment and task to test our methodology. This environment is rarely seen in reality. In our environment, we hand design the reward to be related only to the

feature we assigned to each cell, which is not realistic, as in many real world problems, the reward functions are generally more complex. In the future, we would like to apply this method to more real world problems.

6.3 Threshold Design

In the future, we are also interested in exploring another way of defining our threshold values, especially for β. Currently it is set to be the ratio of the optimal reward value and the lowest reward value, restricting the policy $\hat{\pi}$ to complete certain task, but also puts many pressures on researchers on their abilities in providing very well-defined environments, and termination states that match their expectations on all the baseline requirements. We are thinking about defining it as a scalar value, which lets it stand on its own, and examining whether if it can add more constrains on what kinds of policies are acceptable. We think that this might add more flexibility when applying to problems.

References

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, 09 2004.

[2] S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress, 2020.

[3] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *JMLR Workshop and Conference Proceedings Volume 15: AISTATS 2011*, pages 182–189, Cambridge, MA, USA, Apr. 2011. MIT Press.

[4] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations, 2019.

[5] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences, 2017.

[6] A. Gleave, M. Dennis, S. Legg, S. Russell, and J. Leike. Quantifying differences in reward functions, 2021.

[7] E. C. Grigore, A. Roncone, O. Mangin, and B. Scassellati. Preference-based assistance prediction for human-robot collaboration tasks. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4441–4448, 2018.

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

[9] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei. Reward learning from human preferences and demonstrations in atari, 2018.

[10] S. M. Katz, A. Maleki, E. Biyik, and M. J. Kochenderfer. Preference-based learning of reward function features. *CoRR*, abs/2103.02727, 2021.

[11] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, 05 2000.

[12] M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh. Learning reward functions by integrating human demonstrations and preferences, 2019.

[13] N. Ratliff, J. Bagnell, and M. Zinkevich. Maximum margin planning. pages 729–736, 06 2006.

[14] D. Sadigh, A. Dragan, S. Sastry, and S. Seshia. Active preference-based learning of reward functions. 07 2017.

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

[16] H. Sugiyama, T. Meguro, and Y. Minami. Preference-learning based inverse reinforcement learning for dialog control. In *Proc. Interspeech 2012*, pages 222–225, 2012.

[17] A. Tabrez, S. Agrawal, and B. Hayes. Explanation-based reward coaching to improve human performance via reinforcement learning. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 249–257, 2019.

[18] A. Wilson. A bayesian approach for policy learning from trajectory preference queries. 12 2012.

[19] C. Wirth, R. Akrour, G. Neumann, and J. Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18, 12 2017.

[20] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning, 2016.

[21] J. Zheng, S. Liu, and L. Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. *Proceedings of the National Conference on Artificial Intelligence*, 3:2198–2205, 01 2014.

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