ROBOT LEARNING FROM HETEROGENEOUS DEMONSTRATION

A Dissertation
Presented to
The Academic Faculty

By

Letian Chen

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science
at Georgia Institute of Technology

Georgia Institute of Technology

May 2020

ROBOT LEARNING FROM HETEROGENEOUS DEMONSTRATION

Approved by:

Date Approved: April 22, 2020

The real danger is not that computers will begin to think like men, but that men will begin to think like computers.

Sydney Harris

I dedicate my thesis to my academic advisor, Dr. Matthew Gombolay. Thank you so much for guiding me throughout the process of research and thesis writing.

I also dedicate this thesis to my family, who have always supported me.

I dedicate this thesis to my girlfriend, Zhuoting. You are always my biggest comfort whenever there are storms on my way ahead.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Dr. Matthew Gombolay. He is always there for all of my questions and doubts. He consistently steered me in the right direction whenever I needed it.

I would also like to thank my thesis committee members, Dr. Sonia Chernova and Dr. Harish Ravichandar for their passionate participation and input.

I must express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my life. This accomplishment would not have been possible without them. Thank you.

I want to thank my friends in CORE Robotics Lab. Thank you for all the inspiring discussion. Late-night work was a pleasure with you guys.

I want to thank Yuki, who kept me company during the whole thesis writing process.

I also want to acknowledge my funding sources. This work was sponsored by NASA Early Career Fellowship (80HQTR19NOA01-19ECF-B1), the Office of Naval Research (N00014-18-S-B001), MIT Lincoln Laboratory (7000437192), and institutional funding by the Georgia Institue of Technology.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

Learning from Demonstration (LfD) has become a ubiquitous and user-friendly technique to teach a robot how to perform a task (e.g., playing Ping Pong) without the need to use a traditional programming language (e.g., C++). As these systems are increasingly being placed in the hands of everyday users, researchers are faced with the reality that end-users are a heterogeneous population with varying levels of skills and experiences. This heterogeneity violates almost universal assumptions in LfD algorithms that demonstrations given by users are near-optimal and uniform in how the task is accomplished. In this thesis, I present algorithms to tackle two specific types of heterogeneity: heterogeneous strategy and heterogeneous performance.

First, I present Multi-Strategy Reward Distillation (MSRD), which tackles the problem of learning from users who have adopted heterogeneous strategies. MSRD extracts separate task reward and strategy reward, which represents task specification and demonstrator's strategic preference, respectively. We are able to extract the task reward that has 0.998 and 0.943 correlation with ground-truth reward on two simulated robotic tasks and successfully deploy it on a real-robot table-tennis task.

Second, I develop two algorithms to address the problem of learning from suboptimal demonstration: SSRR and OP-AIRL. SSRR is a novel mechanism to regress over noisy demonstrations to infer an idealized reward function. OP-AIRL is a mechanism to learn a policy that more effectively teases out ambiguity from sub-optimal demonstrations. By combining SSRR with OP-AIRL, we are able to achieve a 688% and a 254% improvement over state-of-the-art on two simulated robot tasks.

CHAPTER 1
INTRODUCTION

Over the last few years, we have seen robotics technology increasingly applied to more and more application scenarios, ranging from everyday household uses [1] and autonomous manufacturing [2] to disaster response [3] and scientific discovery [4]. With robots being used for such a variety of tasks, traditional design of control policies for every task becomes intractable for robot experts. Furthermore, traditional robot programming methods require expertise in coding, a significant time investment, and users to explicitly specify the sequence of actions or movements a robot must execute in order to accomplish the task [5]. Furthermore, each task has numerous variations so that even a manually specified control policy becomes brittle. Moreover, end-users may have their own preferences towards specific tasks. As it is impractical to assume programming skills and robotics expertise for end-users, a natural solution to the aforementioned problems would be Learning from Demonstration (LfD) techniques. LfD facilitates nonexpert robot programming by implicitly learning task constraints and requirements from simply recording demonstrations [5]. It is worth noting that working with novice users is not the only motivation for LfD, some techniques are designed specifically with expert users in mind, including manufacturing and the military [6].

1.1 Learning from Demonstration

We draw upon seminal work in Chernova and Thomaz [6] to state our adopted definition of LfD:

> *Learning from Demonstration (LfD)* explores techniques for learning a task policy from examples provided by a human teacher.

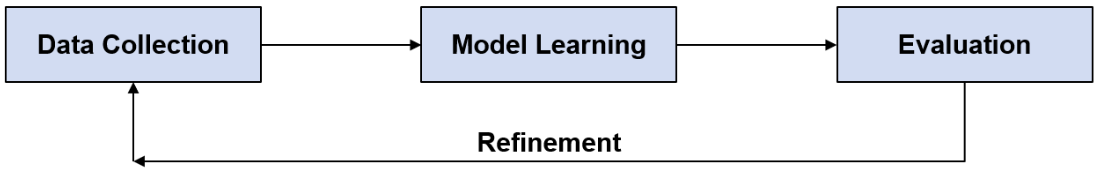


Figure 1.1: Pipeline of Learning from Demonstration

As described in [6], the canonical LfD pipeline includes (illustrated in Figure 1.1)

1. Data collection: a human teacher gives demonstration to the robot via methods such as kinesthetic teaching, teleoperation, or video recording of the human performing the task. The essential key to the data collection phase is to give the robot enough variability for it to adjust to variable environments, while also ensuring the demonstrations clearly achieve the same goal that the human intends to achieve.

2. Model learning: using an expert-designed model and the data collected in the previous phase, machine learning techniques are applied to tune the model to appropriately represent the collected data. Most importantly, machine learning techniques should learn to adapt to the variations in the data and achieve the objective the human wants to accomplish.

3. Evaluation and Refinement: a robot could perform what it has learned and humans could criticize or intervene during the process to provide extra data for it to learn from. Therefore, it could loop back to the data collection step, thus creating a closed-loop tuning method until the performance is satisfying.

In this thesis, I focus on developing novel techniques for the first two steps: 1) data collection and 2) model learning. A key limitation inherent in how this pipeline is typically applied is that it presumes that human demonstrators readily provide demonstrations that are well-suited to train the specific machine learning algorithms to meet the evaluation criteria defined. However, humans typically lack the technical awareness of these algorithms and provide data that are quite difficult for most LfD algorithms to leverage. As such, the focus of this thesis is on designing algorithms that are better able to utilize data from end-users; in particular, the heterogeneity of the demonstrations these users provide. This thesis addresses two key types of heterogeneity: strategy heterogeneity and performance heterogeneity; though there are many to be addressed in future work, as we enumerate in Section 1.2.3.

An essential key to successful machine learning is the abundance of data. Recent machine learning techniques, especially Deep Learning [7], when given moderate assumptions, have the property that more data leads to better performance and that more data beats a cleverer algorithms [8]. However, when facing complex tasks, e.g., complicated sequential decision making, people will have different heuristics because they cannot calculate for the optimal solution (principle of bounded rationality [9]), and thus causing what we call *heterogeneity* in strategies and performance in demonstrations. Continuous control task for robots is indeed one of such preplexing tasks in that when given a goal, there are often many ways to achieve the goal and it is not clear which one is significantly better than the others. For example, tennis is a task that has a single clear goal: return the tennis ball back to opponent's side. Yet, humans have developed tens of different strikes (e.g., backhand, backspin, drop shot, flat, groundstroke, etc.). If we want to let a human teach a robot to play tennis from demonstration, the recorded demonstrations will surely be varied and non-optimal. Previous researches have also identified heterogeneity in human demonstrations. Nikolaidis, Nath, Procaccia, and Srinivasa [10] point out that when giving demonstrations, experts typically adopt heuristics (i.e., "mental shortcuts"[11]) to solve challenging optimization problems, and these highly refined strategies can present a heterogeneity in behavior across task demonstrators. Sammut, Hurst, Kedzier, and Michie [12] found that commercial airline pilots exhibit significant heterogeneity in performing a well-posed task (e.g., executing a pre-specified flight plan) as to make it more practical to learn from a single trajectory and disregard the remaining data.

Thus, despite the long awareness of the heterogeneity problem, few have sought to fix it in a way other than assuming homogeneity or disregarding some of the data. Handling heterogeneity in Learning from Demonstration is an essential step for LfD to embrace big data and deep learning era when it is more and more easy to collect demonstrations from people all over the world, e.g. using Cloud Robotics techniques [13]. In this thesis, I propose several novel machine learning algorithms to better account for the heterogeneity in demonstrations.

1.2 Problem Description - Demonstration Heterogeneity

1.2.1 Formal Problem Description

We formalize demonstration heterogeneity in a probabilistic Markov Decision Process (MDP) framework.

First, we formalize Markov Decision Process. A Markov Decision Process (MDP) M is a 6-tuple $\langle S, A, R, T, \gamma, \rho_0 \rangle$, where S is the state space, and A is the action space. $\gamma \in (0, 1)$ is the temporal discount factor, representing the relative preference for sooner rewards. $R(s, a)$ represents the reward after executing action a in state s. In some cases, $R(s, a)$ could be simplified as $R(s)$. $T(s, a, s')$ is the transition probability to s' from state s after taking action a. $\rho_0(s)$ is the initial state distribution. The standard goal is to find the optimal policy $\pi^* : S \rightarrow A$ that maximizes the discounted future reward

$$\pi^* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi} \left[\sum_{i=0}^{T} \gamma^t R(s_t, a_t) \right]. \tag{1.1}$$

$\tau = (s_0, a_0, \cdots, s_T, a_T)$ denotes a sequence of states and actions induced by the policy and dynamics, and T represents episode length. We instead consider a more general maximum entropy objective introduced by Ziebart [14], which augments the standard objective with an entropy bonus to favor stochastic policies and to encourage exploration during optimization

$$\pi^* = \arg\max_\pi J(\pi) = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=0}^{T}\gamma^t R(s_t,a_t) + \alpha H(\pi(\cdot|s_t))\right]. \tag{1.2}$$

In this thesis, we assume we have access to robot state-action pairs, which means the data collection method could be either kinesthetic teaching or teleoperation, but to make it compatible with learning from observation, we have to make additional system assumptions, such as in [15]. Under such setting, we have a dataset of demonstrated trajectories, $\mathcal{D} = \{\tau_1, \tau_2, \cdots, \tau_N\}$. Each τ_i consists of a sequence of states and actions, $\tau_i = (s_{i,0}, a_{i,0}, s_{i,1}, a_{i,1}, \cdots, s_{i,T}, a_{i,T})$. In homogeneous demonstration, we could assume all trajectories $\tau_i \ \forall i \in \{1..N\}$ come from a single policy π_e interacting with environment, i.e., $a_t \sim \pi_e(s_t), s_{t+1} \sim T(s_t, a_t)$.

In this case, estimating the expert policy π_e by maximizing the likelihood of trajectories results in a maximum likelihood problem on each state:

$$\begin{aligned}\arg\max_\pi P(\tau) &= \arg\max_\pi \prod_{i=1}^{T}\pi(a_i|s_i)T(s_{i+1}|s_i,a_i)\\ &= \arg\max_\pi \prod_{i=1}^{T}\pi(a_i|s_i)\end{aligned} \tag{1.3}$$

If given enough state-action pair generated by π_e, the estimated π could be arbitrarily close to π_e (consistency property of maximum likelihood estimator).

However, in a heterogeneous demonstration setting, trajectories come from different policies $\pi_i \ \forall i \in \{1, 2, \cdots, N\}$, in which N is the number of different policies. We could denote a different dataset corresponding to each demonstration policy $\mathcal{D}^{(i)} = \{\tau_1^{(i)}, \tau_2^{(i)}, \cdots, \tau_M^{(i)}\}$, $\tau_j^{(i)} \sim \pi_i$, where $i \in \{1, 2, \cdots, N\}$ is strategy index, and M is the number of demonstration trajectories for one policy. If we treat all $\mathcal{D}^{(i)}$ as coming from a single policy, the maximum likelihood estimator result could be the mean of all the policies, which could be far away from any expert policy.

In other words, if we model the conditional probability distribution $P(a|s)$ or the joint distribution $P(s, a)$, homogeneous demonstrations will present a single-mode pattern while heterogeneous demonstration will present multi-mode pattern. Therefore, it will be hard for a single-mode model -as typical Deep Learning techniques are- to achieve density matching in heterogeneous Learning from Demonstration. The single-mode model will either achieve mode-seeking (converge to one mode and ignore all other modes) or coverage-seeking (cover all modes while putting its own mode in the middle where density is actually low for any expert model). More discussion is provided in Section 2.7.

1.2.2 An Illustrative Example

In this section, we show a simple but illustrative heterogeneous demonstration example. We consider a navigation (path planning) task for a robot to move from top-left corner to bottom-right corner, as shown in Figure 1.2. To reach the goal, the robot could follow π_1 and go right and then go down. It could also follow π_2 and go down first and then go right. If the task is just to arrive at the star as fast as possible, both trajectories are optimal and both could present in human demonstration. State-action joint densities of the two trajectories are clearly different.

1.2.3 Demonstration Heterogeneity Category

Demonstration heterogeneity could come from several different sources:

1. Heterogeneous strategies: different policies have similar performance. E.g., example given in Section 1.2.2, different strikes in tennis.

2. Heterogeneous performance levels: different policies have variable performance. Some or even all demonstrations are suboptimal. E.g., a naïve tennis player and an expert tennis player giving demonstrations in the same time.

3. Heterogeneous environments: policies are semantically similar, but the environment

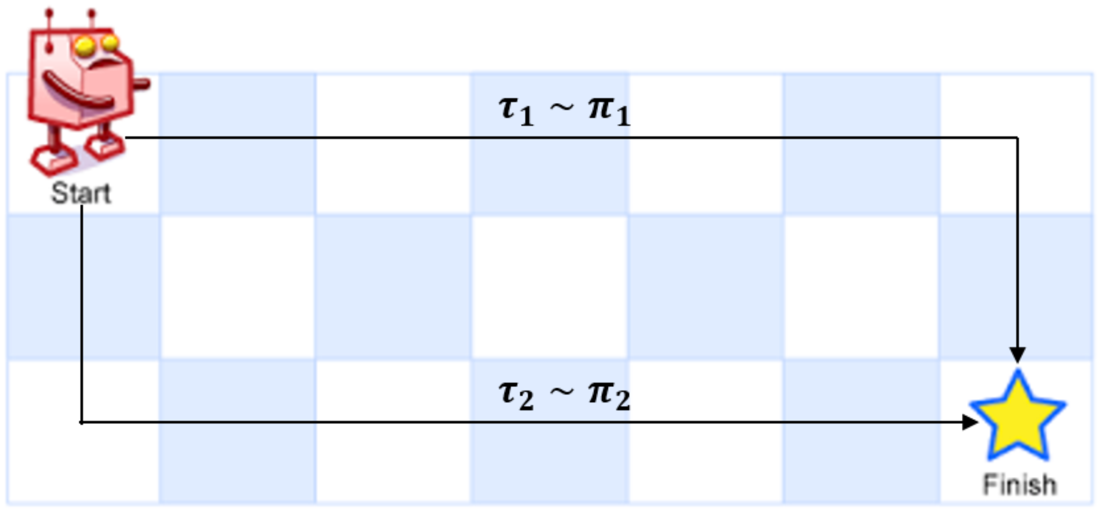


Figure 1.2: An illustrative example to show heterogeneous demonstration. Figure adopted from Leetcode [16].

state representation is different. E.g., one type of household robot in two different homes trying to cook coffee.

4. Heterogeneous embodiments: policies are semantically similar, but demonstrations are on different robots (which could have different Degrees of Freedom). E.g., two different types of household robot trying to cook coffee.

5. Heterogeneous tasks: the tasks demonstrated are different, but the tasks are related, or the skills required for the tasks are similar. E.g., two identical household robots in the same home and one is trying to cook coffee while the other is trying to brew some tea.

Each type of heterogeneity listed above would, in general, have differing downstream implications for any LfD algorithms.

In this thesis, we look into the first two heterogeneity sources, i.e., heterogeneous strategies and heterogeneous performance levels, and leave the latter three heterogeneity for future work. We argue the first two types of heterogeneity are the most common within offline demonstrations, while the latter three are essential keys to cloud robotics where demonstrations for different environments, different embodiments, and different tasks could be meshed together for a universal learning.

1.3 Thesis Contributions

In this thesis, I present work to tackle two most common sources of heterogeneity: strategy heterogeneity and performance heterogeneity.

I start with strategy heterogeneity in Chapter 2, where I propose the Multi-Strategy Reward Distillation (MSRD) method to better extract the task specification from heterogeneous strategies. As a bonus, we could also extract each demonstrator's preference function to better understand the user's preference, which could be further used to deliver user-preference-aware service. I test the algorithm on both simulated and real robots and illustrate the accuracy of the task and strategy reward recovered from heterogeneous strategy demonstrations.

In Chapter 3, I propose two novel algorithms named Optimality-Parameterized Adversarial Inverse Reinforcement Learning (OP-AIRL) and Self-Supervised Reward Regression (SSRR) to tackle the heterogeneous performance demonstrations. I test the two algorithms on two simulated robotic domains and show that both algorithms not only learn from different performance-level demonstrations, but also achieve better performance on the task than the best suboptimal demonstration.

CHAPTER 2
MULTI-STRATEGY REWARD DISTILLATION

2.1 Introduction

In this chapter, we make the assumption that heterogeneous strategy policies $(\pi_1, \cdots, \pi_N)$ is a result of different reward functions $(R_1, \cdots, R_N)$ that different policies are optimizing.

We present four contributions related to the main algorithm for this Chapter: Multi-Strategy Reward Distillation (MSRD) [1]. First, we propose MSRD, a novel IRL framework where we jointly learn task reward and strategic rewards and tease out heterogeneity to gain a better estimation of the task reward and strategic reward component of each strategy. Second, we show MSRD's success on two virtual robot control tasks and one real-world physical robot table tennis task. Third, the results indicate the MSRD's learned task reward function achieves high correlation with the ground-truth task reward and the learned strategy rewards also achieve high correlation with the ground-truth strategic preferences. Fourth, we also develop a method that helps generate heterogeneous policies of different strategies in simulated environments, and thus alleviates burden of expensive heterogeneous human demonstration collection when testing algorithms. This technique is leveraged by our virtual experiments to synthesize heterogeneous task strategies. Moreover, by this way we are able to access the ground-truth strategic preference and evaluate how well MSRD recovers the strategy reward while humans typically have difficulty describing their underlying strategic preferences.

We begin in Section 2.2 with a review of prior work. In Section 2.3 we introduce preliminaries that help understand MSRD, followed by details of MSRD in Section 2.4. Section 2.5 shows the experimental setup in two simulated robot control tasks and one

[1]This work was published by ACM on 15th ACM/IEEE International Conference on Human-Robot Interaction [17].

real-world physical robot table tennis task. In Section 2.6, we show MSRD's effectiveness empirically on both the simulated and real domains. Finally, in Section 2.7, we discuss the results of MSRD and point out some limitations that motivates future work.

2.2 Related Work

Imitation learning (IL) is a straightforward but popular choice of LfD algorithm. IL works by directly learning a predictive model which takes current state as input and outputs action. However, the learned policy is highly intertwined with environment dynamics. Slight changes in dynamics or highly stochastic environments could cause IL to easily fail due to a "covariate" shift [18].

In contrast, the demonstrated reward function is a more transferable and robust definition of the task, representing latent objectives a behavior tries to accomplish. Inverse Reinforcement Learning (IRL) is an LfD approach that aims to infer such demonstrator's objective (i.e., reward function) given a set of performed trajectories. Effective reward learning holds utility even after environment dynamics change [19].

IRL suffers from two major fundamental shortcomings [20]. First, IRL is known to be an ill-posed problem as there are infinitely many reward functions that could explain expert demonstration as optimal including degenerated cases (e.g., $\mathbf{R} = 0$ [21]). Two primary methods exist in the IRL literature that aim to solve this ambiguity: maximum margin approaches [22, 23] and probabilistic approaches [24, 25, 14]. Maximum margin approaches try to find a reward function that explains an expert's trajectory not only as being optimal, but also as being better than all other trajectories by a margin. Probabilistic approaches (e.g. Bayesian IRL [24], Maximum-Entropy IRL [25], and Maximum Causal Entropy IRL [14]) assume trajectories with higher rewards have an exponentially higher probability of being generated and apply a maximum likelihood framework to solve for such a reward function. Advantages of this framework include its tolerance of non-optimal demonstrations and ease of applying gradient-based optimization. Maximum-entropy based IRL methods are thus suitable for deep neural network models to find the reward function [26], obviating the need for feature engineering. Despite the advantages of a maximum entropy IRL (ME-IRL) framework, perfect information about the system dynamics and the ability to calculate the exact state feature counts are required. Guided Cost Learning (GCL [27]) and its successor, Adversarial Inverse Reinforcement Learning (AIRL [28]) have been proposed to tackle the unknown dynamics problem. AIRL poses the reward learning problem in an adversarial setting, where the reward function tries to assign a high reward to expert demonstrations and assign a low reward to generated trajectories. Meanwhile, the policy is optimized over the learned reward function to maximize its expected reward. AIRL also made strides to disentangle reward functions from the environment's dynamics (less reward shaping).

Despite all the efforts made in prior work to address the problem of reward ambiguity in IRL, one of the most effective solutions is still simply to collect more data. By collecting more data, one might hope to more readily tease out the signal (i.e., the desired task) from the noise (i.e., inter-demonstrator variance not necessary to task completion). However, as we introduced in Chapter 1, more data may in fact bring about more heterogeneity (illustrated in Figure 1.2 [16]), particularly when humans focus on demonstrating task in multiple ways as a form of teaching the domain of feasible task executions. The typical approach is to assume homogeneity over demonstrators; however, Sammut, Hurst, Kedzier, and Michie [12] found extreme heterogeneity in flight route across pilots as to make it more practical to learn from a single trajectory and disregard the remaining data.

Previous work in IRL has tried to address heterogeneity explicitly [10], but only focused on simplified examples which do not typify the complexity of the real world robot tasks. We are motivated by such settings to develop a method that might still be capable of generating a robust task specification while also leveraging all available data to help mitigate the curse of dimensionality. Another line of work that tries to model heterogeneous expert data lies in imitation learning. Generative Adversarial Imitation Learning (GAIL) [29] is a popular imitation learning algorithm in which a discriminator tries to distinguish between expert trajectories and generated trajectories, while a generator tries to deceive the discriminator. Some extensions to GAIL utilize a latent variable to model multi-modal (multi-strategy) demonstration data [30, 31]. Despite the good performance of imitation learning algorithms in some contexts, such methods are still severely sensitive to a change of environments (e.g., a small disturbance could result in total failure of intended task). By contrast, the reward function is more robust to a change of dynamics in the environment, as the function represents the intention of the behavior.

MSRD is also related to preference learning, as we explicitly extract a strategic preference component from the reward function. Preference learning is a well-explored research area that aims to learn individuals' proclivities. In robotics, preference learning typically involves learning a human's preference and adapting a robot's behavior to better collaborate with human-beings [10, 32]. There are several methods that could infer preference information without asking the human to explicitly provide preference information [33, 34]. For example, Schafer, Frankowski, Herlocker, and Sen [35] utilizes collaborative filtering for a recommender system and Xu, Ratner, Dragan, Levine, and Finn [36] formulates a meta-IRL problem that could learn a prior over preferences. A more recent direction of preference learning lies in the multi-task setting. Dimitrakakis and Rothkopf [37] models reward-policy pairs (R_i, π_i) drawn from an unknown prior. Choi and Kim [38] integrates Dirichlet process mixture model into Bayesian IRL as a task prior. Repeated inverse reinforcement learning (RIRL) [39] formalizes the setting in which a user is observed performing different tasks. The goal of RIRL is to infer a task-independent preference. However, one of the biggest limitations of RIRL is the method assumes full knowledge of each task's reward function and the relative weights between task rewards and preference rewards, which is unrealistic in most real-world applications. Observational Repeated IRL (ORIRL) relaxes the assumption by introducing a learnable relative weight yet still assumes perfect knowledge of the task reward Woodworth, Ferrari, Zosa, and Riek [40] . Furthermore, both RIRL and ORIRL work is carried out in discrete finite state-action space, which is a significant simplification of real-world, continuous, state-action space. While preference learning in a multi-task setting tries to learn shared preferences across tasks, we propose to learn a shared task reward from observing different strategies for the same task.

Few previous work have addressed multiple-reward-function learning. One such approach by Nikolaidis, Gu, Ramakrishnan, and Shah [41] clusters different kinds of behaviors and applies inverse reinforcement learning on each cluster. Another approach, Option GAN [42], learns a division in demonstration state space and a separate reward function and policy for each subspace. However, neither considers the relationship between the learned reward functions. In contrast, our work poses a common task reward function in each of the reward functions, together with a separate strategy reward function for each strategy, which enables us to tease out task reward and strategy rewards.

2.3 Background

2.3.1 Inverse Reinforcement Learning

IRL considers an MDP sans reward function ($M\backslash R$) with the goal being to infer reward function $R(s,a)$ given a set of demonstrated trajectories $\mathcal{D} = \{\tau_1, \tau_2, \cdots, \tau_N\}$. A typical assumption for IRL is that demonstrated trajectories are optimal, or at least near-optimal. In the maximum entropy IRL (Max-Ent IRL) framework, inference of the reward function is turned to a maximum likelihood optimization problem by assigning occurrence probability of a trajectory proportional to the exponential of discounted cumulative reward, $p_\theta(\tau) \propto e^{\sum_{t=0}^{T} \gamma^t r_\theta(s_t,a_t)}$. Therefore, Max-Ent IRL aims to find the reward function under which the demonstrated trajectories have the highest likelihood, as shown in Equation 2.1.

$$\max_\theta \mathbb{E}_{\tau\sim\mathcal{D}}[\log p_\theta(\tau)] \tag{2.1}$$

AIRL [28] casts the optimization in a generative adversarial framework, learning a discriminator, D, to distinguish between experts and a deceiving generator, π, that learns to imitate the expert. This framework follows Max-Ent IRL's assumption that trajectory likelihood is proportional to the exponential of rewards. D is defined in Equation 2.2, where $f_\theta(\tau)$ is the learnable reward function and π is the current policy. D is updated by minimizing its cross entropy loss to distinguish expert trajectories from generator policy rollouts (Equation 2.3). π is trained to maximize the cumulative pseudo-reward function given by $f(s, a)$.

$$D_\theta(s,a) = \frac{\exp(f_\theta(s,a))}{\exp(f_\theta(s,a)) + \pi(a|s)} \tag{2.2}$$

$$L_D = -\mathbb{E}_{\tau\sim\mathcal{D},(s,a)\sim\tau}[\log D(s,a)] - \mathbb{E}_{\tau\sim\pi,(s,a)\sim\tau}[1 - \log D(s,a)] \tag{2.3}$$

2.3.2 Neural Network Distillation

Neural network distillation applies supervised regression to train a student network to produce the same output distribution as a trained teacher network. The method was first proposed by Hinton, Vinyals, and Dean [43] and has been applied in RL mainly for performing policy distillation [44, 45, 46]. Particularly, Teh, Bapst, Czarnecki, Quan, Kirkpatrick, Hadsell, Heess, and Pascanu [45] proposes that instead of distilling each policy π_i to a general policy π_0, we could gain a faster convergence with a two-column architecture by defining $\pi_i = \pi_0 + \widetilde{\pi}_i$. Accordingly, π_i only needs to learn a near-zero difference between a common policy and the task-specific policy.

2.4 Algorithm

As mentioned in Section 2.2, reward function is a more transferable and robust definition of the task. Therefore in MSRD we transform the heterogeneity on different policies to heterogeneity on the different rewards that the policies are trying to optimize. In the space of reward functions, the common parts across heterogeneous demonstrations are more clear: achieving the intended goals has high rewards.

We consider a setup in which there is only one task (one MDP $M = (S, A, R^{(0)}, T, \gamma, \rho_0)$), but the demonstrations are generated by employing varying strategies. Different strategies may come from different experts who have personalized strategical preferences or one expert that has mastered several. Therefore, despite heterogeneity within demonstrations, all trajectories in the dataset should still be near-optimal in terms of task reward $R^{(0)}$. We denote each strategy's demonstration dataset as $\mathcal{D}^{(i)} = \{\tau_1^{(i)}, \tau_2^{(i)}, \cdots, \tau_M^{(i)}\}$, where $i \in \{1, 2, \cdots, N\}$ is strategy index, M is the number of demonstration trajectories for one strategy, and N is the number of strategies. Our first objective is to infer a shared task reward function $R^{(0)}$ despite there being different strategies in the demonstration dataset. The second objective is to infer the strategy-only reward $\widetilde{R^{(i)}}$. Combining strategy-only reward with the task reward function will result in a strategy-combined reward $R^{(i)}$, which should induce the observed expert strategical behaviors.

2.4.1 Task and Strategy Reward

We first propose to model the strategy-combined reward function that is optimized by a demonstrator to be a linear combination of the task and the strategy-only reward as given by Equation 2.4.

$$R^{(i)}(\cdot) = R^{(0)}(\cdot) + \alpha_i \widetilde{R^{(i)}}(\cdot) \tag{2.4}$$

Despite the simplicity, we argue Equation 2.4 makes a reasonable assumption. Several previous IRL works also apply linear, if not more constraining, assumptions to combine reward functions: Amin, Jiang, and Singh [39] create a combined reward by adding a task reward with a cross-task shared preference, while we add shared task reward with each strategy reward; Woodworth, Ferrari, Zosa, and Riek [40] propose a similar formulation, except their task reward is known and they try to learn task-independent preference in a multi-task setting. In the RL literature, there is also substantial work combining rewards linearly (e.g., adding intrinsic reward, such as curiosity or entropy to the original task reward for the sake of exploration). In fact, many engineered reward functions are also a linear combination of several reward components (e.g., for OpenAI Gym [47] MuJoCo [48] hopper environment, its reward function is defined as a linear combination of forward speed, living bonus, and action penalty). Additionally, assuming a linear combination could also provide interpretability (see Figure 2.3). Our approach is unique in that it shares the task reward while having the flexibility on each strategy reward, which we argue is more realistic while allowing us to apply joint inference of the task reward and strategy reward.

2.4.2 Reward Network Distillation

To infer the shared task reward function between different strategies, we propose utilizing network distillation to distill common knowledge from each separately learned strategy-combined reward $R^{(i)}$ to the task reward function $R^{(0)}$. We also want to regularize $R^{(i)}$ to be close to $R^{(0)}$, since we have the prior knowledge that despite heterogeneous strategic preferences, all experts should still be prioritizing optimizing the task reward $R^{(0)}$ to achieve at least near-optimal performance. Previous distillation methods mainly focus on distilling classification results, and therefore KL-divergence between teacher and student outputs could be a good choice for regularization. However, reward functions are real-valued functions and therefore probabilistic distance metrics do not fit. Thus, we propose to regularize the expected L2-norm of the difference between the reward functions, as shown in Equation 2.5, in which $\pi^{(i)}$ is the optimal policy under reward function $R^{(i)}_{\theta_i}$.

$$L_{\text{reg}} = \mathbb{E}_{(s,a)\sim\pi^{(i)}}\left(\left|\left|R^{(i)}_{\theta_i}(s,a) - R^{(0)}_{\theta_0}(s,a)\right|\right|_2\right) \tag{2.5}$$

Note that we are using an index both on θ and R to denote that each strategy-combined reward $R^{(i)}$ has its own reward parameters, and that these are approximated by separate neural networks with parameters θ_i for each strategy and θ_0 for the task reward. There is no parameter sharing between strategy and task reward.

Due to the computational cost of optimizing $\pi^{(i)}$ using RL, we seek to avoid fully optimizing it inside the IRL loop. Therefore, we apply an iterative reward function and policy training schedule, similar to AIRL. Through such way, we could learn reward function and policy simultaneously. Combining AIRL's objective (Equation 2.3) with the distillation objective, we want to maximize L_D in Equation 2.6.

$$\begin{aligned} L_D = \sum_{i=1}^{N} \Big[& \mathbb{E}_{(s,a)\sim\tau_j^{(i)}\sim\mathcal{D}^{(i)}} \log D_{\theta_i}(s,a) \\ & + \mathbb{E}_{(s,a)\sim\pi^{(i)}} \log(1 - D_{\theta_i}(s,a)) \\ & - \mathbb{E}_{(s,a)} \left(\left\| R^{(i)}_{\theta_i}(s,a) - R^{(0)}_{\theta_0}(s,a) \right\|_2 \right) \Big] \end{aligned} \tag{2.6}$$

D is dependent on θ_i via Equation 2.2 ($R^{(i)}_{\theta_i}$ corresponds to f_θ). Each $\pi^{(i)}$ optimizes[2] $R^{(i)}_{\theta_i}$. Yet, while Equation 2.6 should be able to distill the shared reward into $R^{(0)}_{\theta_0}$, the distillation is inefficient as $R^{(0)}_{\theta_0}$ will work as a strong regularization for $R^{(i)}_{\theta_i}$ before successful distillation.

Instead, our proposed reward structure in Equation 2.4 allows for a two-column re-parameterization, speeding up knowledge transfer and making the learning process easier [45]. Combining Equation 2.4 and Equation 2.6, we arrive at Equation 2.7.

$$\begin{aligned} L_D = \sum_{i=1}^{N} \Big[& \mathbb{E}_{(s,a)\sim\tau_j^{(i)}\sim\mathcal{D}^{(i)}} \log D_{\theta_i,\theta_0}(s,a) \\ & + \mathbb{E}_{(s,a)\sim\pi^{(i)}} \log\left(1 - D_{\theta_i,\theta_0}(s,a)\right) \\ & - \alpha_i \mathbb{E}_{(s,a)} \left(\left\| \widetilde{R^{(i)}_{\theta_i}}(s,a) \right\|_2 \right) \Big] \end{aligned} \tag{2.7}$$

The key difference between Equation 2.7 and Equation 2.6 is that D depends on both $R^{(0)}_{\theta_0}$ and $\widetilde{R^{(i)}_{\theta_i}}$ instead of separate $R^{(i)}_{\theta_i}$. Thus, $R^{(0)}_{\theta_0}$ directly updates from the discriminator's

[2]We choose Trust Region Policy Optimization (TRPO) [28]

loss rather than waiting for knowledge to be learned by a strategy-combined reward and subsequently distilled into a task reward. Further, the last term of Equation 2.7 reduces to a simple L2-regularization on strategy-only reward's output, weighted by α_i. This formulation provides us with a new view to interpret the relative weights of the strategy-only reward α_i: the larger α_i is, the more the strategy-only reward will influence the strategy-combined reward. Therefore, we will have higher regularization to account for possible overwhelming of the task reward function. Comparing Equation 2.7 and 2.3, we could interpret MSRD in another view: optimizing θ_i only via IRL objective results in a combination of task and strategy reward, and adding regularization on strategy reward will encourage to encode only necessary information in θ_i and share more knowledge in θ_0.

2.4.3 Multi-Strategy Reward Distillation

We summarize our algorithm in Algorithm 1. In the algorithm we first collect the heterogeneous-strategy expert dataset and initialize network parameters as well as relative weights. For each training epoch, we will iterate over all strategies (line 5). For each strategy, we first collect K trajectories generated by its corresponding policy $\pi^{(i)}$ (line 6). We also sample expert trajectories for strategy i from the dataset (line 7). We then train the Discriminator (reward function) with loss given by Equation 2.7 and data τ_j^{gen} and τ_j^{exp} (line 8). After training the reward function, we could assign pseudo-reward to trajectories we generate with reward function $R_{\theta_i}^{(i)}$ (line 9). Finally, we update the policy according to the trajectory generated and pseudo-reward signal (line 10). In practice, we could also postpone the gradient update for $R^{(0)}$ at the end of one sweep of strategies to stabilize the learning of the task reward.

2.5 Experiment Setup

We tested MSRD on both virtual and real-world environments. Here, we describe our experiments, showing results in Section 2.6.

Algorithm 1 Multi-Strategy Reward Distillation

```
 1: Obtain heterogeneous-strategy expert dataset $\mathcal{D}^{(i)} = \{\tau_1^{(i)}, \cdots, \tau_M^{(i)}\}\ \forall i \in \{1, \cdots, N\}$
 2: Initialize $R^{(0)}$, $R^{(i)}$, $\pi^{(i)} \quad \forall i \in \{1, 2, \cdots, N\}$
 3: Determine relative weights $\alpha_i \quad \forall i \in \{1, 2, \cdots, N\}$
 4: while not converged do
 5:     for i = 1 to $N$ do
 6:         Collect $K$ ($K \leq M$) trajectories $\tau_j^{\text{gen}}$ by executing $\pi^{(i)}$
 7:         Sample $K$ trajectories $\tau_j^{\text{exp}}$ from $\mathcal{D}^{(i)}$
 8:         Train $\theta_i$ and $\theta_0$ with Equation 2.7, $\tau_j^{\text{gen}}$ and $\tau_j^{\text{exp}}$
 9:         Assign reward for all transitions in $\tau_j^{\text{gen}}$: $r(s, a) = R_{\theta_i}^{(i)}(s, a)$
10:         Update policy $\pi_i$ using trajectories $\tau_j^{\text{gen}}$ via TRPO with entropy bonus to encourage exploration
11:     end for
12: end while
13: return $R^{(0)}$, $R^{(i)}$, $\pi^{(i)}$
```

2.5.1 Virtual Experiments

We first tested MSRD on two simulated environments: a simpler inverted pendulum control task and a more difficult hopper locomotion task (see Figure 2.5). The goal of the inverted pendulum task is to balance a pendulum on a cart by moving the cart left/right, making it a single degree of freedom (DoF) problem, based on a 2D observation. The reward for inverted pendulum is defined as the negative absolute value of the pendulum angle from upright position. The objective of hopper is to control 3-DoF joints to move forward based on its 11-dimensional observation. The reward for hopper is defined as the speed at which it moves forward. We used the OpenAI Gym [47] MuJoCo [48] implementation for both environments but made the following changes to fit our application: 1) remove termination judgements to gain flexibility in behaviors; 2) add timeout constraint of 1,000 steps.

Heterogeneous Demonstration Collection

Our algorithm can utilize heterogeneous-strategy demonstrations. Therefore, we first need to generate a variety of demonstrations to emulate heterogeneous strategies that humans will apply in solving problems for our virtual experiments. Typical RL algorithms can only generate single-modal policy for each task in attempt to maximize the task reward. Some previous work has tried to generate diverse behaviors [49, 50]. Among them, Diversity is All You Need (DIAYN) trains a discriminator to distinguish different behaviors and trains policies utilizing the discriminator's output as a pseudo-reward. The pseudo-reward is shown in Equation 2.8, where z is the strategy index, q is the posterior decoding z from s, and $p(z)$ is the prior distribution. However, DIAYN only discovers different behaviors without task specification, so we augment DIAYN to incorporate task reward by the linear form of Equation 2.4.

$$r_z(s,a) = \log q_\phi(z|s) - \log p(z), \tag{2.8}$$

We also propose a method to encourage different strategies taking different actions in the same state via a diversity reward in which k is the strategy index and π_k is the policy for strategy k (Equation 2.9). Equation 2.9 encourages the KL-divergence between different strategy's policies to be large. Linearly combining KL-encouraged diversity reward with the task reward, we can train strategies that optimize both the task goal and the diversity goal.

$$r^k_{\text{KL}}(s) = \sum_{i=1}^{N} KL(\pi_k(\cdot|s)||\pi_i(\cdot|s)), \tag{2.9}$$

We trained both “DIAYN + Extrinsic Reward” and ”KL-Encouraged + Extrinsic Reward” policies to collect heterogeneous trajectories that applied different strategies to solve the task. From all the strategies generated, we chose 20 strategies for inverted pendulum and the two most significant strategies for hopper. Generally, different strategies in inverted pendulum encourage the cart to stay at different angles, but some strategies maintain dynamic balance by periodically moving the cart left and right. Two different strategies in hopper are ”Hop” and ”Crawl” as illustrated in Figure 2.5.

2.5.2 Real-World Experiments

For our second environment, we tested MSRD on its capability to learn various table tennis strokes from expert human demonstration. Participants were asked to kinetically teach a robot arm to hit an incoming ping pong ball using three different stroke strategies: push, slice, and top spin, from both a forehand and backhand position.

The setup of our table tennis environment consists of a 7-degree of freedom Sawyer robot arm from Rethink Robotics, two Go-Pro Hero7 cameras, and a Newgy Robo-Pong 2055 ball feeder. The angle and speed the ball feeder is set to was calculated empirically previous to collecting human demonstrations and kept constant throughout our experiment.

To communicate with Sawyer, we used Rethink Robotic's Intera SDK that integrates seamlessly with the Robot Operating System (ROS) middleware. We record the participant's demonstrations by subscribing to the robot joint state topic. This provides us with joint position and velocity at a rate of of 100 Hz.

For our vision system, we used the GoPro Hero7 Black to track the ping pong ball's trajectory throughout the demonstrations. We chose these cameras for their high frame rate. ElGato Camlinks were used as a connection bus to convert readily stream image data into the computer. This permits us a streaming rate of 60 Hz. To estimate the ping pong ball 3D position in real-world, we chose the triangulation method with a parallel axis camera set up of stereo vision. We utilized OpenCV [51] Library to detect the ball using carefully designed HSV values and contour size. We utilize ping pong ball transition dynamics alongside the calculated location of the ping pong ball in a Extended Kalman Filter to produce an accurate estimate of the ping pong ball's location throughout a trajectory.

Experiment Design and Subjects

We adopt a within-subjects design for demonstration collection, requiring subjects finish all six combinations of positions and strategies. We pseudo-randomized the order of forehand and backhand, as well as the order of three strategies. We recruited 10 subjects from a population of college graduate students. Subjects were compensated.

Experiment Procedure

When the participants arrived, they read and signed a consent form detailing the purpose, duration, and study procedure. For each trial, we began by showing a video tutorial on how to move the robot arm to hit the incoming ball with a specific strategy. After the tutorial, we allowed participant to practice with the automatic ping pong feeder. Once the subject felt ready, we began the recording process. Saved trajectories were those in which the participant was able to successfully return the ball to the opponent's side and the movement of striking closely resembled the strategy assigned. We collected three recordings for each condition. We record at each timestep of the trajectory the robot joint angles, angle rates and the ball position.

2.6 Results

In this section, we report and analyze MSRD's results on three environments and benchmark against AIRL to elucidate MSRD's advantage on recovering both the latent task reward (the essential goal of the demonstrators) and the means by which the task is accomplished (i.e. the strategy). We explore two hypotheses:

H1: *The task reward learned by MSRD has a higher correlation with the true task reward than AIRL.*

H2: *Strategy-only reward learned by MSRD has a higher correlation with true strategic preferences than AIRL.*

We assessed both hypotheses quantitatively and qualitatively for the simulation environments only as the ground-truth reward functions are available. In the physical robot experiment, users are instructed to execute the task instead of optimizing an objective function, meaning that we do not have access to the underlying task reward or strategy-only rewards. Therefore, **H1** and **H2** were assessed qualitatively in the physical robot experi-

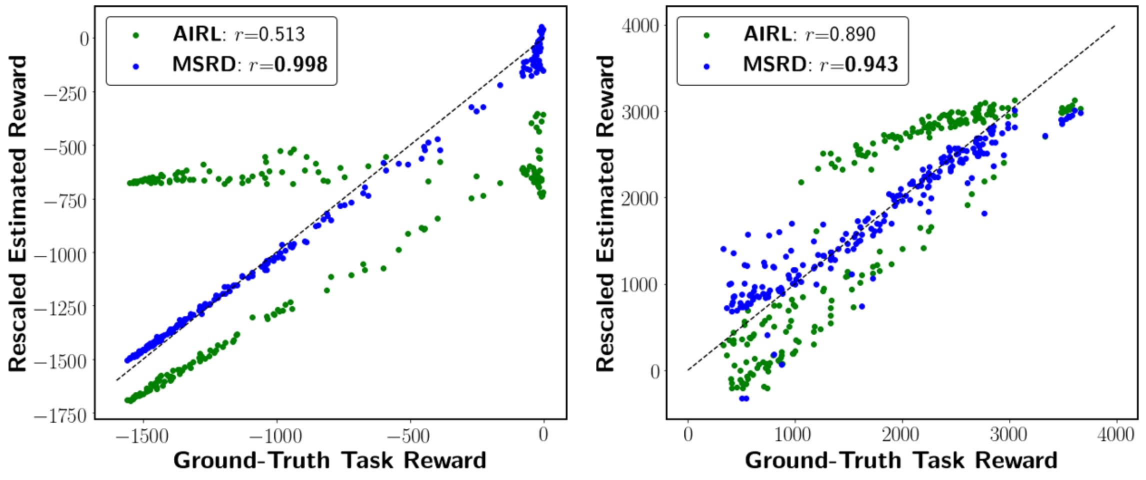


Figure 2.1: Correlation between ground-truth and estimated task reward, normalized for each strategy to $[0, 1]$, for inverted pendulum (left) and hopper (right) environments. Reward is invariant to shift/scale. r is correlation coefficient.

ment. We leverage end-to-end neural network architectures to avoid manual feature extraction. Furthermore, we wanted to provide a fair comparison between methods; given MSRD leverages AIRL, the most appropriate choice is thus AIRL.

2.6.1 Simulated Environments

To test **H1**, we constructed a dataset of trajectories that have various task performances utilizing noise injection [52]. We note that this dataset was also generated with various-strategy policies to be representative of the entire trajectory space. We then evaluated the reward function learned by AIRL and MSRD on the trajectories, comparing estimated vs. ground-truth rewards. We show a correlation of estimated rewards and ground-truth task rewards in Figure 2.1. The task reward function learned through MSRD has a higher correlation with the ground-truth reward function (0.998 and 0.943) versus AIRL (0.51 and 0.89) for each domain, respectively). AIRL's reward function overfits to some strategies and mixes the task reward with that strategy-only reward, making its estimation unreliable for other strategies' trajectories.

To test **H2**, we calculated the correlations of MSRD's strategy-only rewards with the true strategic preferences and compared that with the correlation of AIRL's rewards when

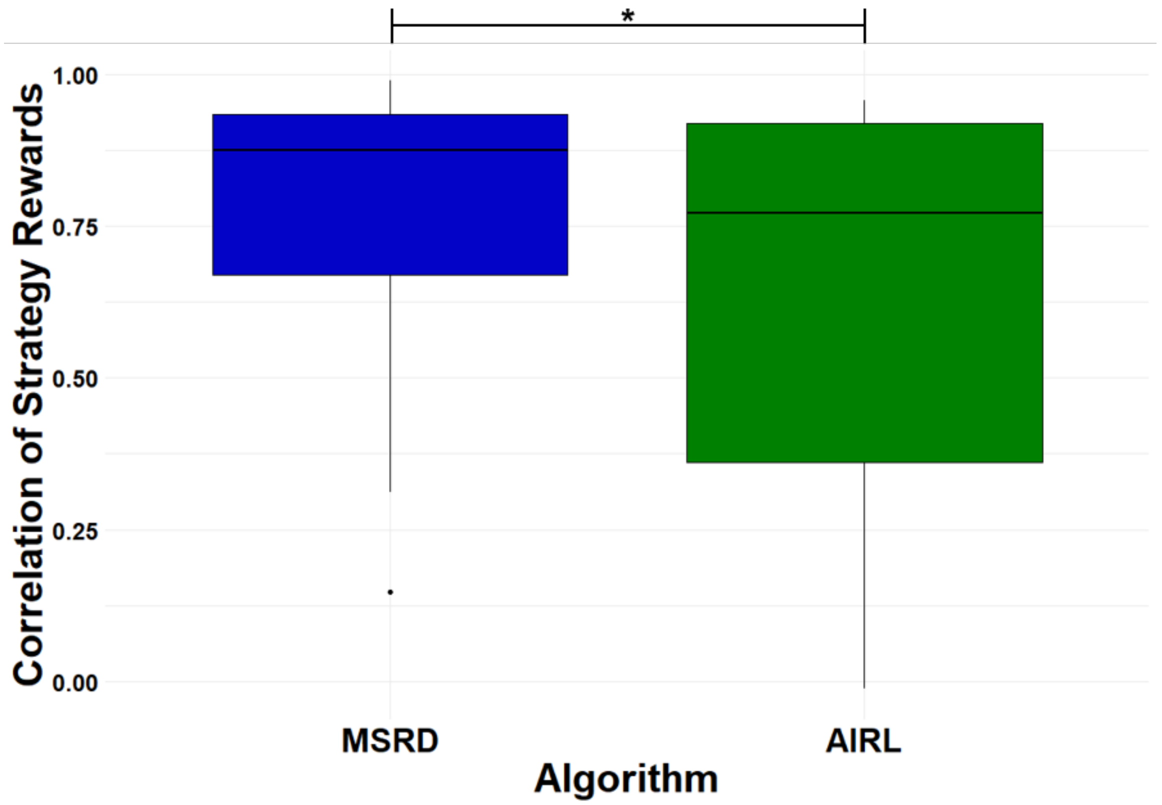


Figure 2.2: Correlation between ground-truth vs. estimated strategy reward by MSRD and AIRL on Inverted Pendulum.

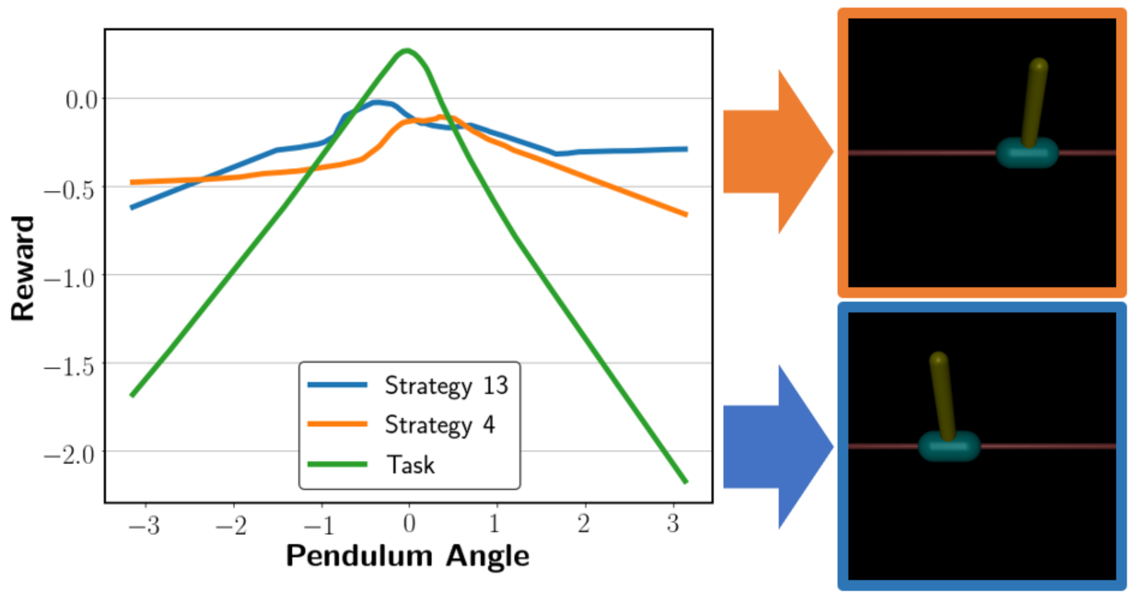


Figure 2.3: Task/Strategy Reward Functions of Inverted Pendulum vs Pendulum Angle and Corresponding Behaviors.

AIRL is trained on each individual strategy. In simulated domains, true strategic preferences are available as the pseudo-reward in Equation 2.8 and 2.9. Correlations of both methods for all strategy rewards in inverted pendulum are shown in Figure 2.2. A paired t-test shows that MSRD achieves a statistically significantly higher correlation (M = 0.779, SD = 0.239) for the strategy rewards versus AIRL (M = 0.635, SD = 0.324) trained separately for each strategy, $t(19) = 1.813$, $p = 0.0428$ (one-tailed). A Shapiro-Wilk test showed the residuals were normally distributed ($p = 0.877$). For the hopper domain, MSRD achieved 0.85 and 0.93 correlation coefficient for the hop and crawl strategy, compared with AIRL's 0.80 and 0.82 respectively. We omit a t-test here due to the limited number of strategies. We could test the discrimination of strategy rewards by evaluating each strategy's reward function on each strategy's trajectory; we expect to observe that the strategy-only reward function of each strategy gives its corresponding trajectory the highest reward. We show in Figure 2.4 that this expectation holds. Out of 20 strategy-only rewards, 16 receive highest rewards in corresponding trajectories. A Binomial test shows we are significantly better than chance ($p < .001$).

We are unable to examine the reward landscape in the inverted pendulum environment as it is four dimensional. Thus, we choose to fix three dimensions (cart position, cart velocity and pendulum angular velocity) to zero and investigate the reward change within the one remaining dimension (pendulum angle). The relationship between rewards and pendulum angles in task and strategy reward functions are illustrated in Figure 2.3, in which the task reward function reaches its peak when the angle of the pendulum is near zero. This precisely recovers the task reward. For strategy-only reward functions, strategy 13 encourages the pendulum to lean left (demonstration behavior shown in bottom-left of Figure 2.3), while strategy 4 encourages the policy to tilt the pendulum right (demonstration behavior shown in bottom-right). Therefore, strategy-only rewards learned by MSRD captures specific preferences within demonstrations. Figure 2.3 also shows the magnitude of the task reward is larger than the strategy reward, which affirms our expectation that an emphasis is being put towards accomplishing the task.

In the hopper environment, it is harder to visualize the reward landscape due to a high-dimensional observation space and a lack of interpretability of states. Therefore, instead of visualizing a reward curve, we evaluate the estimated strategy-only reward on trajectories from both strategies to provide evidence for **H2**. Figure 2.5 shows that when given a hopping trajectory, the hop strategy-only reward function gives higher reward for that behavior than crawl strategy-only reward function. Similarly, in the crawl trajectory case (Figure 2.5), the crawling strategy-only reward gives a higher value than the hop strategy-only reward. Therefore, the strategy-only reward function recovered by MSRD gives a higher reward to the corresponding behavior than the other strategy-only reward function, thus providing encouragement to the policy towards the intended behavior (**H2**).

These results across our simulated environments show our algorithms' success in both task reward recovery and strategy reward decomposition. This capability is a novel contribution to the field of LfD in that we are able to tease out strategies from the underlying task and effectively learn policies that can reproduce both the strategy and a well-performed

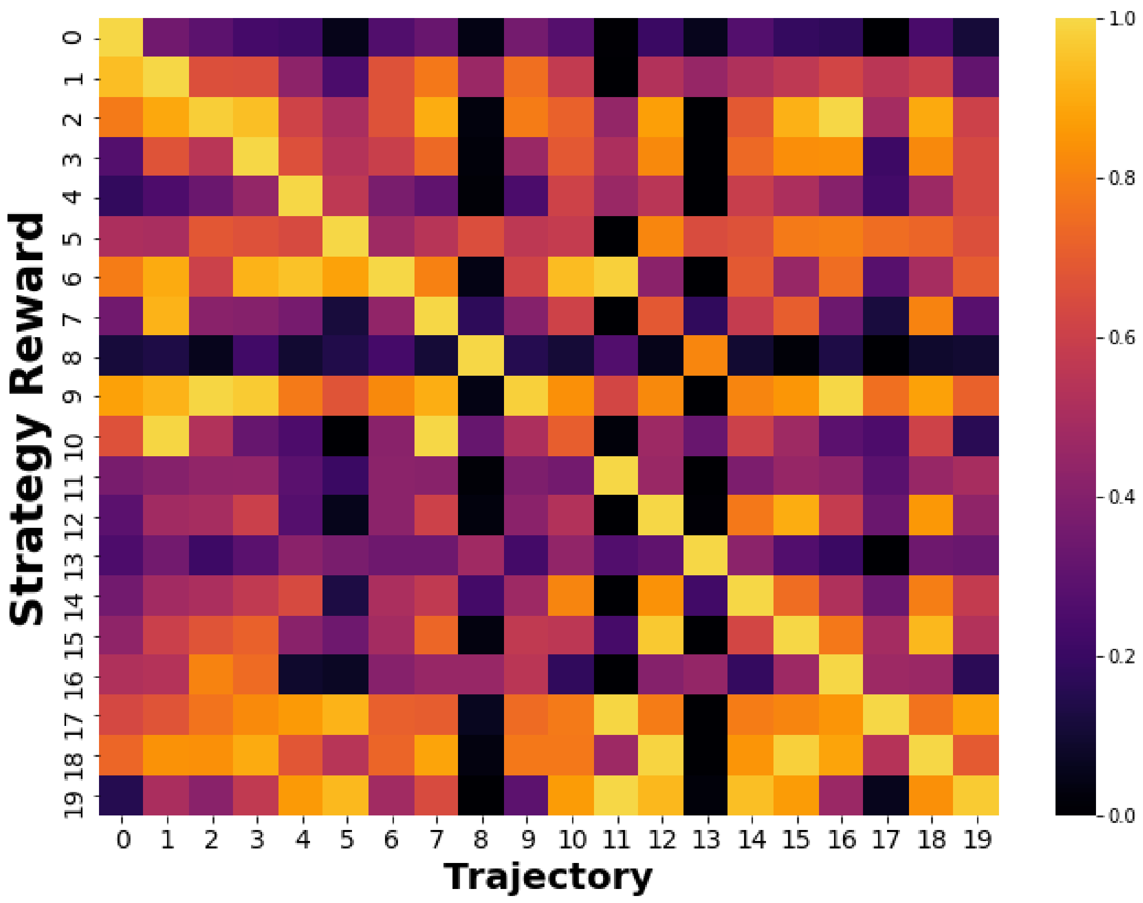


Figure 2.4: Evaluation of strategy rewards on strategy trajectories (Inverted Pendulum); rows normalized to $[0, 1]$.

policy for the underlying task.

2.6.2 Physical Environment

Each strategy was trained for approximately 30 minutes, totaling to 1.5 hours for the three strategies (push, topspin, slice). However, lingering balls that remain on the table due to collision with the net need to manually removed. Also, balls need to be picked up and reloaded into the feeder after every 2-4 iterations (each training iteration consists of six strikes, two of each type). This process resulted in a total training time of 2.5 hours.

We utilize four deep neural networks (DNNs) consisting of three fully connected layers (32 nodes on each hidden-layer) to represent task, push, slice and topspin rewards. The

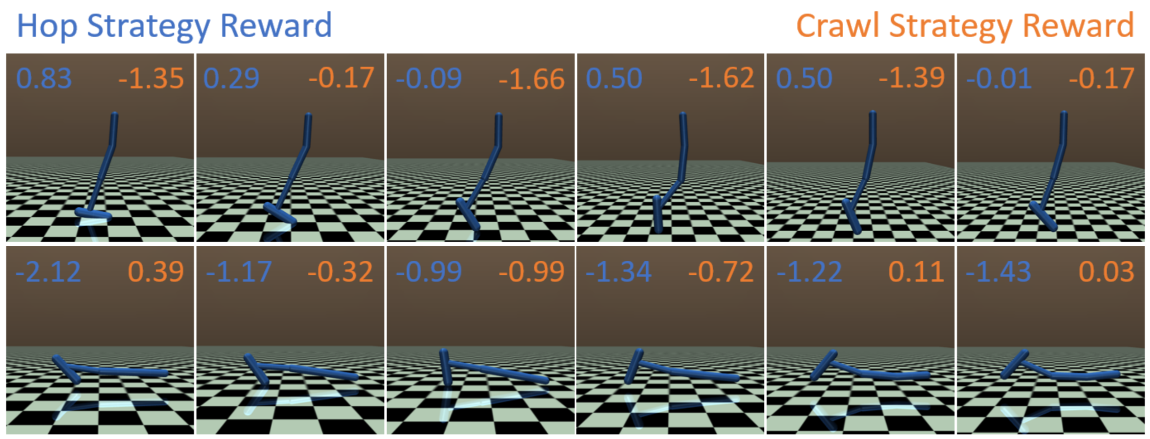


Figure 2.5: Hop and crawl strategy reward on Hop (top) and Crawl (bottom) trajectories. Blue and orange numbers are learned Hop and Crawl strategy rewards, respectively.

ball states alongside robot arm joints serve as our inputs. The label of different types of demonstration (forehand-push, backhand-slice, etc.) is available to our algorithm. Figure 2.6 shows four frames of the learned trajectories for our defined strategies. The change in angle and the upward/downward motion of the paddle throughout the policy trajectory are key factors in the display of different strategies (as these induce spin). Push is associated with a small change in angle as it is not attempting to add any spin onto the ball. Slice places a backspin on the ball, and thus the angle of the paddle will quickly tilt up as shown in Figure 2.6. Conversely, topspin places a topspin on the ball; to do so, the associated trajectory has a quick upward motion. Figure 2.7 provides quantitative evidence that the strategy-only reward should be maximal given a demonstration utilizing the corresponding strategy. After just 30 minutes of training on each strategy, the robot was able to learn to strike 83% of the fed balls. The robot learned to perform all strategies, and the robot's best strategy, topspin, resulted in 90% of balls returned successfully.

To further verify that our task reward was learning the correct behavior, the task reward function was used to evaluate each of the demonstrated trajectories (three trajectories for each of the three strategies). In the case of the original demonstrations (i.e., where the ball was struck and landed in bounds), the average and standard deviation of the task reward across demonstrations and strategies was 1.476 ± 0.051. We then virtually manipulated the

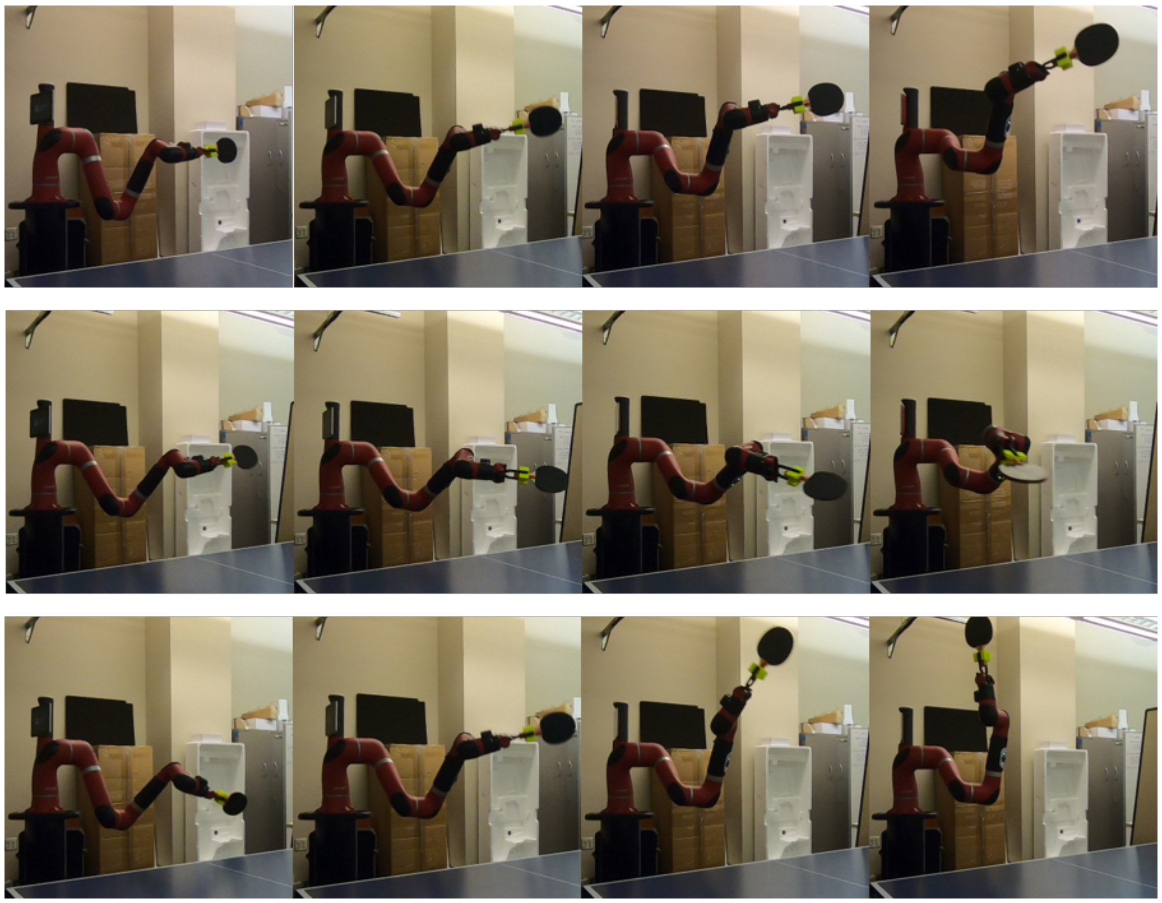


Figure 2.6: Different strategies learned in robot table tennis task. Top to bottom: Push, Slice, and Topspin.

trajectories to indicate that the ball was unsuccessfully returned, which achieved an average and standard deviation for the task reward of only 1.284 ± 0.042. A Friedman Test shows this result is statistically significant ($\chi^2(1, 9) = 9, p < 0.05$), providing support that our task reward is learning to identify successful returns, as unsuccessful trajectories should be associated with lower reward when compared to successful. In this real-world robot control task, we see that MSRD can successfully recover all three strategies' behaviors using human data.

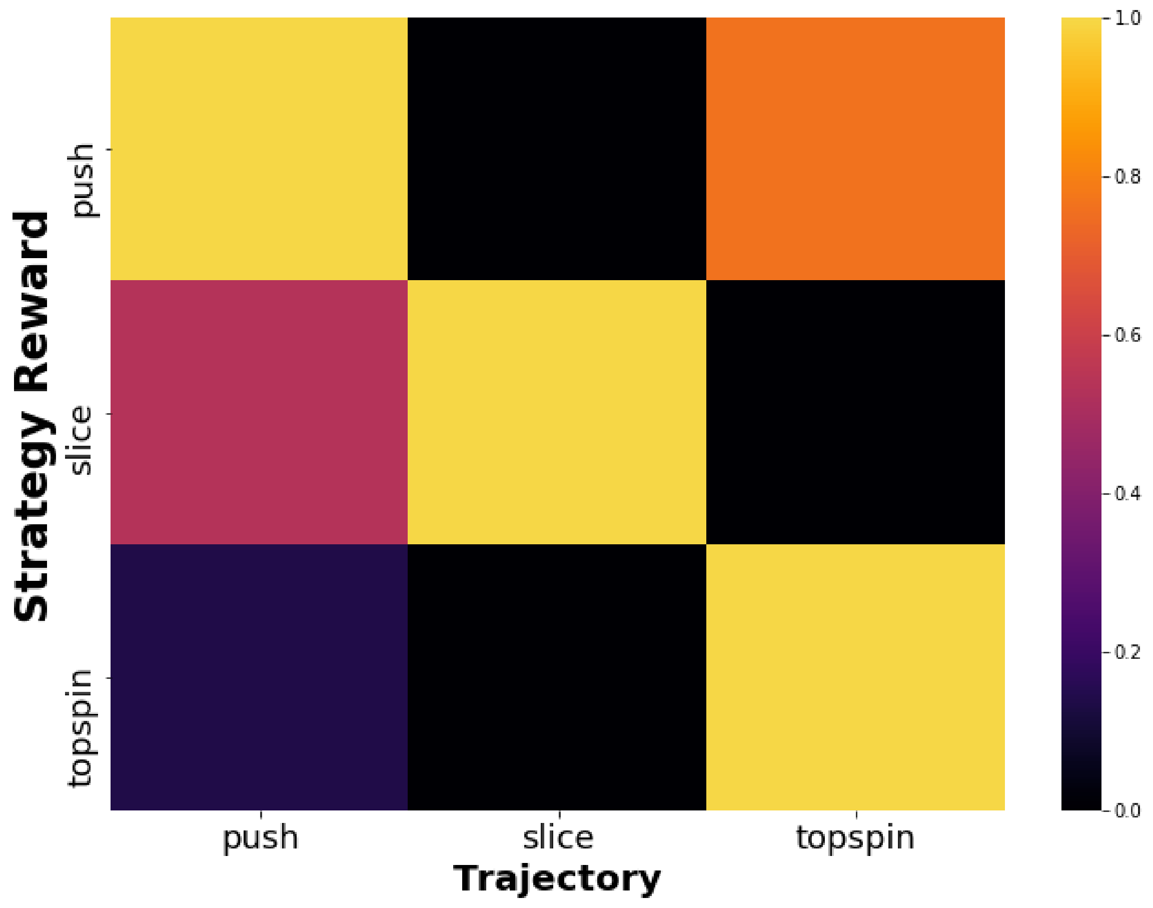


Figure 2.7: Evaluations of Strategy Rewards on Strategy Trajectories for Table Tennis. Each row is normalized to $[0, 1]$.

2.7 Discussion and Future Work

2.7.1 Discussion

As I have discussed in Section 1.2.1, state-action pairs of heterogeneous demonstration illustrate multiple-mode pattern. Like most GAN-based IRL methods, AIRL's reward and policy try to do mode seeking and ignore all other modes, which is illustrated in Figure 2.1. AIRL's mode seeking behavior is exactly the reason why it could mimic a movement very well, compared with coverage-seeking methods such as direct imitation learning [53]. However, a mode-seeking reward is biased and the policy could be a mixture of the underlying task and demonstrator's personal preference. MSRD, on the other hand, cover all the modes with all strategy rewards and the task reward, while also providing the mode-seeking possibility with each individual strategy reward for each strategy. Therefore, it has near-perfect recovery of the task reward as well as good policies that could achieve the task with different strategies.

2.7.2 Future Work

One clear drawback of MSRD is that it requires an explicit labelling for which strategy a demonstration lies in. In some tasks it is very easy for a human to help identify the correct labelling (hop or crawl). However, sometimes demonstrations could be a mixture of several strategies, and a numerical weighting of each strategy is a non-trivial task for a human being. In the future work, one could to utilize different strategy inference methods (i.e., to infer the strategy label) to allow for unlabeled demonstrations and then apply MSRD with the labels. One could also design methods that dynamically adapt the relative weighting of each strategy reward for each demonstration while also learning for the reward functions, making it possible to let two processes adjust to each other.

CHAPTER 3
HETEROGENEOUS-PERFORMANCE REWARD LEARNING

3.1 Introduction

In this chapter, we turn our attention to the second type of heterogeneity in demonstration, heterogeneous-performance levels (see explanation in Section 1.2.3), to "automatically" determine which demonstration is the best and try to learn to be close to it or, even better, achieve higher performance than it. To accomplish this aim, we again argue that learning a reward function that can discriminate between higher or lower performance is a promising approach. As long as we are able to recover an accurate reward function from the suboptimal demonstrations, we would then be able to apply Reinforcement Learning methods to optimize such reward function and obtain an optimal policy, as shown initially in [54, 52].

Despite the large number of IRL methods introduced in Section 2.2, almost all of them assume the optimality of demonstrations to some extent. Maximum margin approaches require absolute optimality of the demonstration, while probabilistic approaches such as maximum-entropy IRL and Bayesian IRL relaxes it to near-optimal. They infer the reward function based on the principle that the given demonstrations have high or the highest return (or high probability if trajectories probability are proportional to the return) among all the possible trajectories. This principle does not hold for the case of completely suboptimal demonstrations. However, the real learning signal that optimal demonstrations give to IRL is that demonstration is **better** than all other trajectories (better meaning higher cumulative ground-truth rewards), which is technically a binary classification signal that we could possibly replicate in the case of suboptimal demonstration. Although we do not know a trajectory that has the highest reward among all the trajectories, we could have access to some relative relationship between trajectories, e.g., one demonstration is better than another or how much is one demonstration better than another. Such relationship information could be accessed by querying humans [54] or automatically generated [52]. Although demonstrations are suboptimal, such relationships are mostly true and thus could lead us to an accurate reward function.

Following such thoughts, we propose Self-Supervised Reward Regression (SSRR), and further combine it with another novel approach, Optimality-Parameterized Adversarial Inverse Reinforcement Learning (OP-AIRL). We demonstrate the success of both methods in two simulated robotic domains, and achieve an even better performance when combining both.

We begin in Section 3.2 with a introduction of a closely related previous work, Disturbance-based Reward Extrapolation (D-REX). In Section 3.3, we introduce our Self-Supervised Reward Regression (SSRR) algorithm to accomplish reward learning from suboptimal demonstrations, followed by Section 3.4 showing its success on two simulated robot control tasks. In Section 3.5, we further introduce a novel data generation method for SSRR called Optimality-Parameterized Adversarial Inverse Reinforcement Learning (OP-AIRL) substituting AIRL in original SSRR. We show that with OP-AIRL, SSRR works even better and achieve a much better performance than the best demonstration in Section 3.6. Finally, in Section 3.7 we conclude the contributions and point out some limitations that motivates future work.

3.2 Preliminaries

Our method draws inspiration from Disturbance-based Reward Extrapolation, D-REX [52], which is a method to automatically generate ranked demonstrations and then learn from them. D-REX works by taking suboptimal demonstrations $\{(s_t, a_t)\}$ and applying behavior cloning on them first:

$$\pi_{BC} = \arg\max_{\pi} \prod \pi(a_t|s_t). \tag{3.1}$$

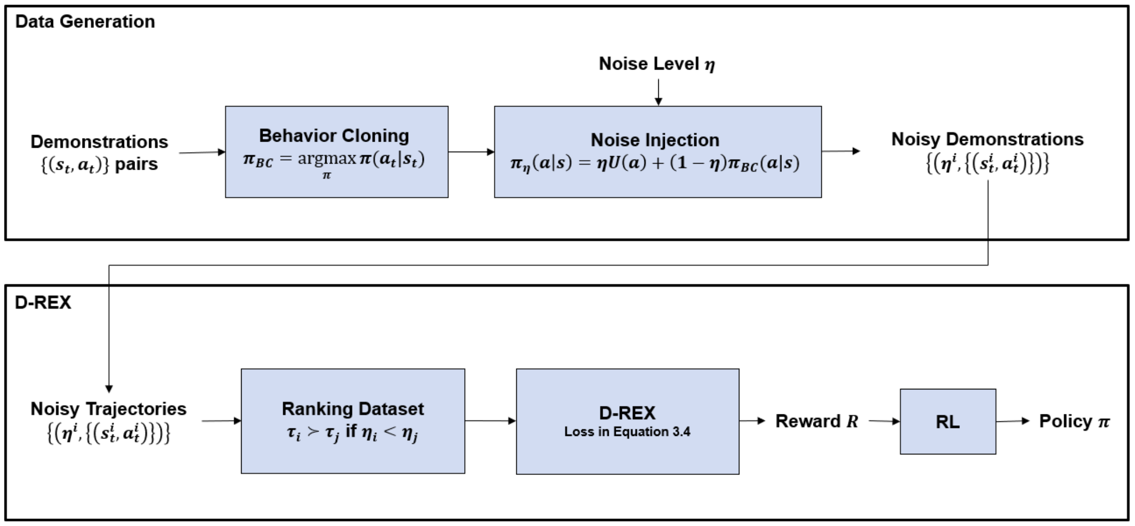

Data Generation
Noise Level η
Demonstrations $\{(s_t, a_t)\}$ pairs
Behavior Cloning $\pi_{BC} = \operatorname{argmax}_{\pi} \pi(a_t|s_t)$
Noise Injection $\pi_\eta(a|s) = \eta U(a) + (1-\eta)\pi_{BC}(a|s)$
Noisy Demonstrations $\{(\eta^i, \{(s_t^i, a_t^i)\})\}$
D-REX
Noisy Trajectories $\{(\eta^i, \{(s_t^i, a_t^i)\})\}$
Ranking Dataset $\tau_i \succ \tau_j$ if $\eta_i < \eta_j$
D-REX
Loss in Equation 3.4
Reward R
RL
Policy π

Figure 3.1: Diagram for D-REX

Then we could add noise (uniform sampling of an action) to the learned π_{BC} to get noisy trajectories (noise percentage η):

$$\tau_\eta \sim \pi_\eta(a|s) = \eta U(a) + (1-\eta)\pi_{BC}(a|s). \tag{3.2}$$

$U(a)$ represents uniform distribution over the entire action space.

Via such noisy trajectory generation method, we could obtain trajectories associated with different noise levels $\eta \in [0, 1]$. D-REX makes the assumption that the higher the noise level is, the lower the trajectory should be ranked, i.e.,

$$\forall \eta_i > \eta_j, R(\tau_{\eta_i}) < R(\tau_{\eta_j}). \tag{3.3}$$

R represents cumulative reward calculation from trajectory $R(\tau) = \sum_{i=0}^{T} \gamma^t r_t$.

D-REX learns the reward directly through supervised learning on the ranking, using a pairwise ranking loss:

$$L(\theta) = -\frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log \frac{\exp \sum_{s\in\tau_i} R_\theta(s)}{\exp \sum_{s\in\tau_i} R_\theta(s) + \exp \sum_{s\in\tau_j} R_\theta(s)}, \tag{3.4}$$

where $\mathcal{P} = \{(i,j) : \tau_i \succ \tau_j\} = \{(i,j) : \eta_i < \eta_j\}$.

In the implementation of D-REX, authors also introduced “snippets”, which are subsampled variant-length consecutive segments from full trajectories, and the ranking pair dataset is based on such snippets instead of full trajectories to increase the data amount. Moreover, D-REX only generates ranked pairs if their noise level is different by at least a margin, i.e., $\eta_i + \epsilon < \eta_j$ and D-REX implementation uses $\epsilon = 0.3$. The process of D-REX is illustrated in Figure 3.1.

3.3 Self-Supervised Reward Regression

3.3.1 D-REX Assumption

We argue D-REX’s loss function does not accurately reflect the noise-performance relationship. It is generally correct to assume higher noise leads to lower rewards, but D-REX’s loss function implicitly adds an assumption for the structure of performances between noise levels. It could be illustrated in a simple 3-noise-level situation.

Theorem 1. *Suppose there are 3 trajectories associated with 3 noise levels, namely (η_1, τ_1), (η_2, τ_2), and (η_3, τ_3), with $\eta_1 < \eta_2 < \eta_3$. Denote the cumulative reward for τ_i as r_i. Then we have $r_2 = \frac{r_1 + r_3}{2}$.*

Proof. We could view the problem as fixing r_1 and r_3, then to optimize r_2 to minimize Equation 3.4.

Denote $R_i = \exp(r_i)$. Thus, we could write the loss

$$\begin{aligned}
L(R_1, R_2, R_3) &= -\log \frac{R_1}{R_1 + R_2} - \log \frac{R_1}{R_1 + R_3} - \log \frac{R_2}{R_2 + R_3} \\
\hat{R_2} &= \arg\min_{R_2} L(R_1, R_2, R_3) \\
&= \arg\max_{R_2} \log \frac{R_1}{R_1 + R_2} + \log \frac{R_1}{R_1 + R_3} + \log \frac{R_2}{R_2 + R_3} \\
&= \arg\max_{R_2} \log \frac{R_1}{R_1 + R_2} + \log \frac{R_2}{R_2 + R_3} \\
&= \arg\max_{R_2} \log \frac{R_1}{R_1 + R_2} \frac{R_2}{R_2 + R_3} \\
&= \arg\max_{R_2} \log \frac{R_1}{R_2 + \frac{R_1 R_3}{R_2} + R_1 + R_3}
\end{aligned} \tag{3.5}$$

Note that $R_2 + \frac{R_1 R_3}{R_2} \geq \sqrt{R_1 R_3}$ and the equation holds and only holds when $R_2 = \sqrt{R_1 R_3}$. Thus,

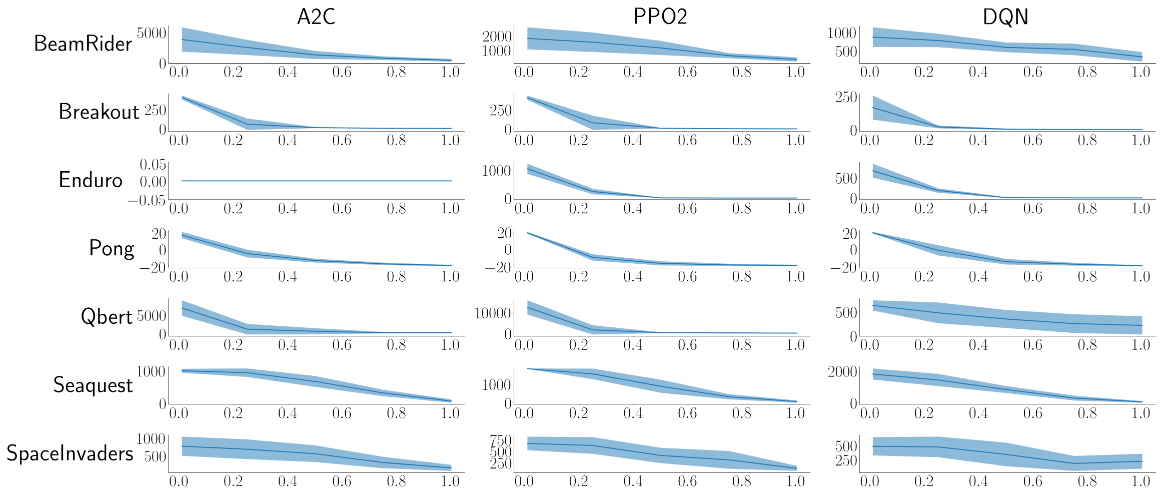
$$\log \frac{R_1}{R_2 + \frac{R_1 R_3}{R_2} + R_1 + R_3} \leq \log \frac{R_1}{R_1 + R_3 + 2\sqrt{R_1 R_3}} \tag{3.6}$$

and the equality holds and only holds when $R_2 = \sqrt{R_1 R_3}$. Therefore, the loss is minimized when $R_2 = \sqrt{R_1 R_3}$, which is equivalent to $r_2 = \frac{r_1 + r_3}{2}$. $\square$

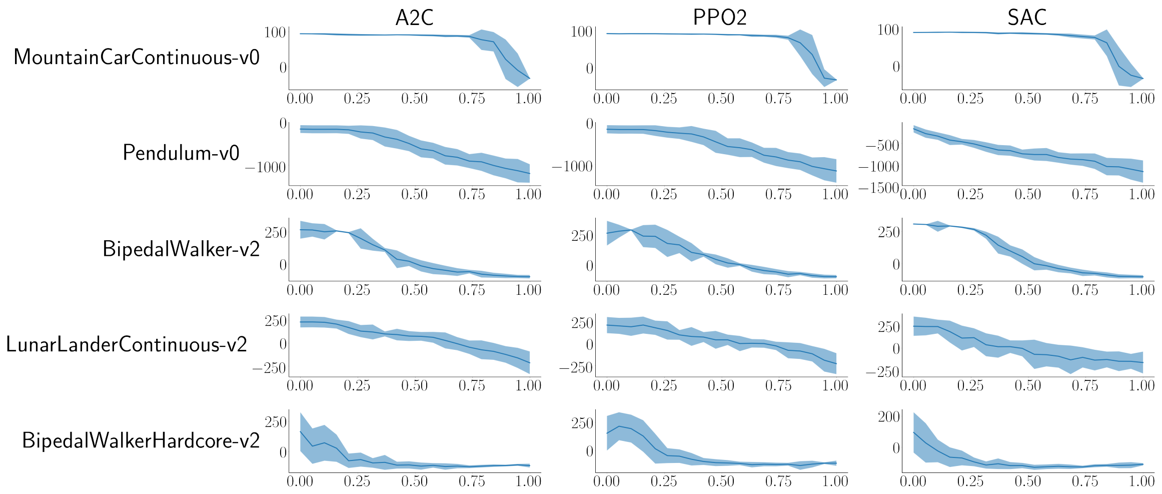
Thus, D-REX's loss function assumes an invariant structure between noise levels across environments or noise options. We show in the next section that this is generally incorrect.

3.3.2 Real Performance-Noise Relationship

We now look into the performance-noise relationship when we have access to the ground-truth reward. Here, we plot such relationship for seven Atari games and five MuJoCo control tasks in Figure 3.2a and Figure 3.2b, respectively. In the figures, row represents different MDP environment, and column represents different policy optimization methods whose optimized policies are injected with noise. In each graph, x-axis represents the noise level, and y-axis represents the ground-truth return. Solid lines are performance



(a) Atari Performance-Noise Relationship



(b) MuJoCo Performance-Noise Relationship

Figure 3.2: Performance Degeneration with Noise Levels Increase on Atari and MuJoCo

averaged over 20 repeats on each noise level, and shadows are standard deviations over the 20 repeats. It is clear that under different environments and different policies, the performance-noise relationships are different and non-linear. D-REX's reward learning approach assumes homogeneous performance-noise relationship across all environments, which is inappropriate as shown.

But how can we do better? An important observation from Figure 3.2a and 3.2b is that the relationship could be fit very well with a 4-parameter sigmoid function:

$$\sigma(\eta) = \frac{c}{1 + \exp(-k(\eta - x_0))} + y_0. \tag{3.7}$$

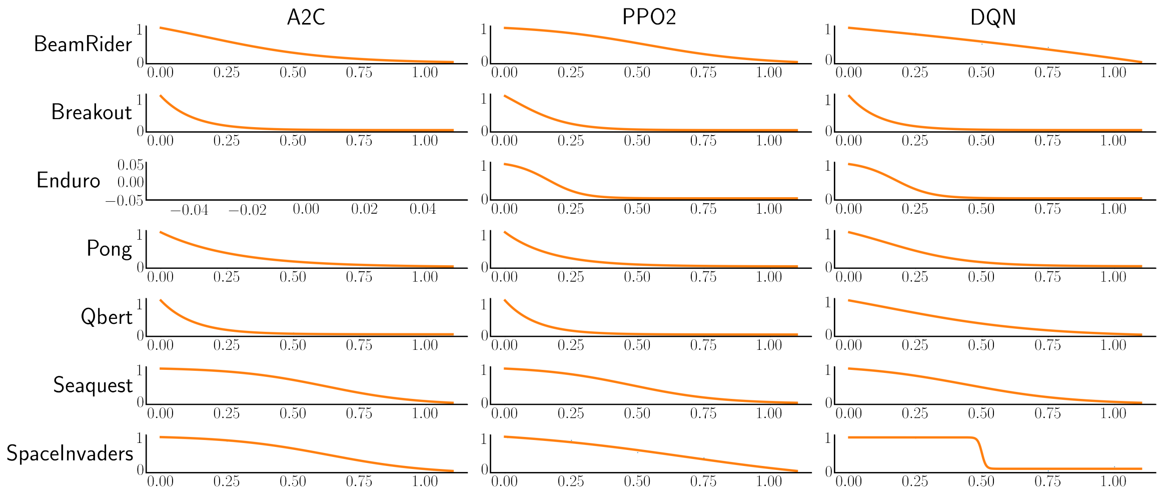
The fitting results are shown in Figure 3.3a and 3.3b.

Therefore, if we could have a way to extract such curve on our heterogeneous-performance demonstrations instead of blindly following the same relationship assumption as D-REX does, the learned reward should be more accurate and also lead to better policy training.

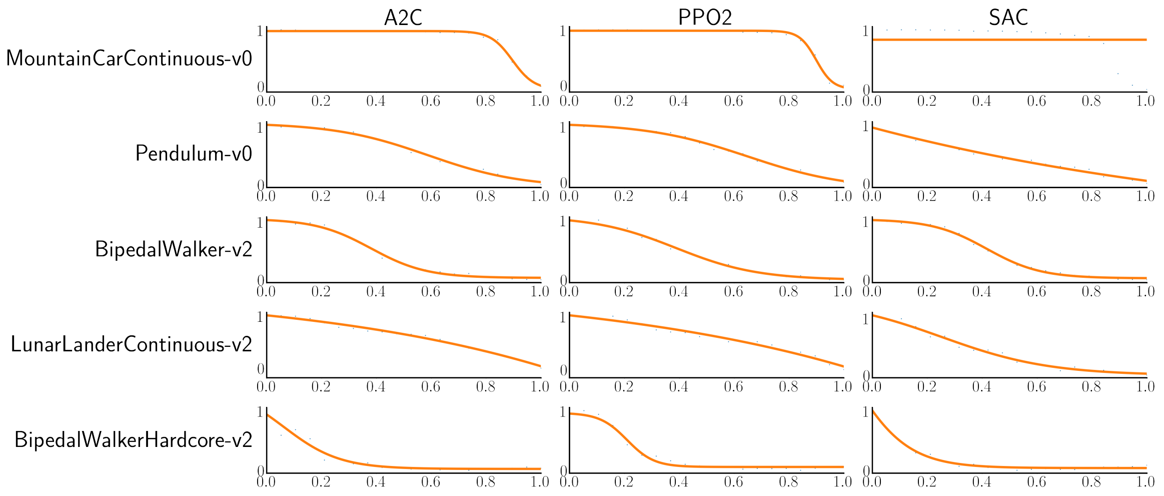
3.3.3 Self-Supervised Reward Regression

As illustrated in Figure 3.4, the idea of Self-Supervised Reward Regression (SSRR) starts with utilizing AIRL on the demonstrations to obtain a initial reward $\tilde{R}$ and a initial policy $\tilde{\pi}$. Similar to D-REX, we inject noises into the learned policy but in addition, we also utilize the initial reward $\tilde{R}$ to criticize the generated noisy trajectories. Thus, the trajectories now consist of four parts: the noise parameter η, state s_t, action a_t, and the corresponding initial reward $\tilde{r_t} = \tilde{R}(s_t, a_t)$.

Since we now have access to the initial reward, we will be able to generate the approximate performance-noise relationship just as what we did for ground-truth reward in Section 3.3.2 and fit a sigmoid function to represent the relationship. We argue that despite the roughness and possible non-smoothness of the AIRL reward (represented by neural network), a sigmoid function only has 4 parameters and is a well-behaving function.



(a) Atari Performance-Noise Relationship Fitted by Sigmoid Function



(b) MuJoCo Performance-Noise Relationship Fitted by Sigmoid Function

Figure 3.3: Performance Degeneration with Noise Levels Increase Sigmoid Fit on Atari and MuJoCo

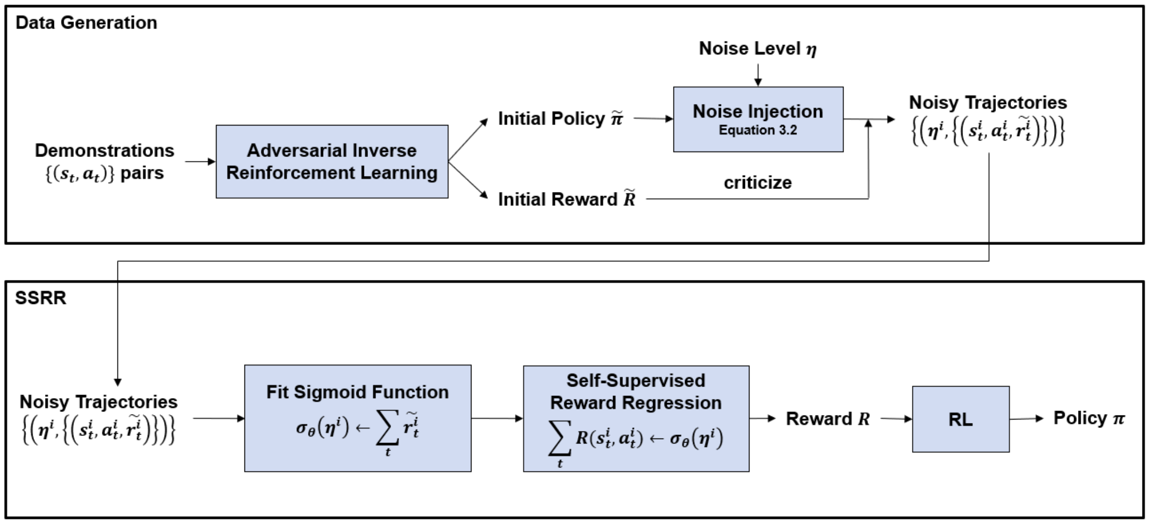

Data Generation
Noise Level η
Demonstrations $\{(s_t, a_t)\}$ pairs
Adversarial Inverse Reinforcement Learning
Initial Policy $\tilde{\pi}$
Noise Injection Equation 3.2
Noisy Trajectories $\{(\eta^i, \{(s_t^i, a_t^i, \tilde{r}_t^i)\})\}$
Initial Reward $\tilde{R}$
criticize
SSRR
Noisy Trajectories $\{(\eta^i, \{(s_t^i, a_t^i, \tilde{r}_t^i)\})\}$
Fit Sigmoid Function $\sigma_\theta(\eta^i) \leftarrow \sum_t \tilde{r}_t^i$
Self-Supervised Reward Regression $\sum_t R(s_t^i, a_t^i) \leftarrow \sigma_\theta(\eta^i)$
Reward R
RL
Policy π

Figure 3.4: Self-Supervised Reward Regression Diagram

Therefore, even if the initial reward is not absolutely accurate, the fitted sigmoid function could filter out high-frequency noises and be precise. Instead of doing the pairwise ranking classification as in D-REX, we now have a good estimation of the corresponding reward value for each noise level, and we could regress the trajectory cumulative rewards to the sigmoid results, i.e. minimizing a mean squared error (MSE) loss:

$$L_{\text{SSRR}} = \mathbb{E}_{\tau^i}\left[\left(\left(\sum_{t=0}^{T} R(s_t^i, a_t^i)\right) - \sigma(\eta^i)\right)^2\right]. \tag{3.8}$$

3.4 Results of SSRR

In this section, we show the learning results of SSRR in two simulated robotic task domains: HalfCheetah and Hopper. The environment setup is the same as in Chapter 2 (MuJoCo implementation). In our results below, we show a comparison of how well our approach is able to infer the ground-truth reward function versus D-REX. By learning a more accurate reward function, we are laying groundwork to then learn a policy that can more readily transcend the quality of the initial human demonstration.

Table 3.1: Learned Reward Correlation Coefficients with Ground-Truth Reward Comparison between SSRR and D-REX on HalfCheetah (Noisy Training data generated by Behavior Cloning)

Repeat	BC + D-REX	BC + SSRR
1	0.902	0.917
2	0.859	0.936
3	0.917	0.889
4	0.933	0.933
5	0.866	0.912
Mean (Std. Dev.)	0.900 (0.025)	0.929 (0.017)

Table 3.2: Learned Reward Correlation Coefficients with Ground-Truth Reward Comparison between SSRR and D-REX on HalfCheetah (Noisy Training data generated by AIRL)

Repeat	AIRL + D-REX	AIRL + SSRR
1	0.830	0.911
2	0.804	0.962
3	0.810	0.943
4	0.832	0.935
5	0.814	0.912
Mean (Std. Dev.)	0.818 (0.011)	0.933 (0.019)

3.4.1 HalfCheetah

Similar to Chapter 2, we compare the learned reward function's correlation with ground-truth reward between SSRR and D-REX. Across all the experiments on HalfCheetah in this Chapter, the original demonstration is just a single demonstration that has 187.7 undiscounted cumulative rewards, which is highly suboptimal as an optimal trajectory typically reaches more than $2,000$ rewards.

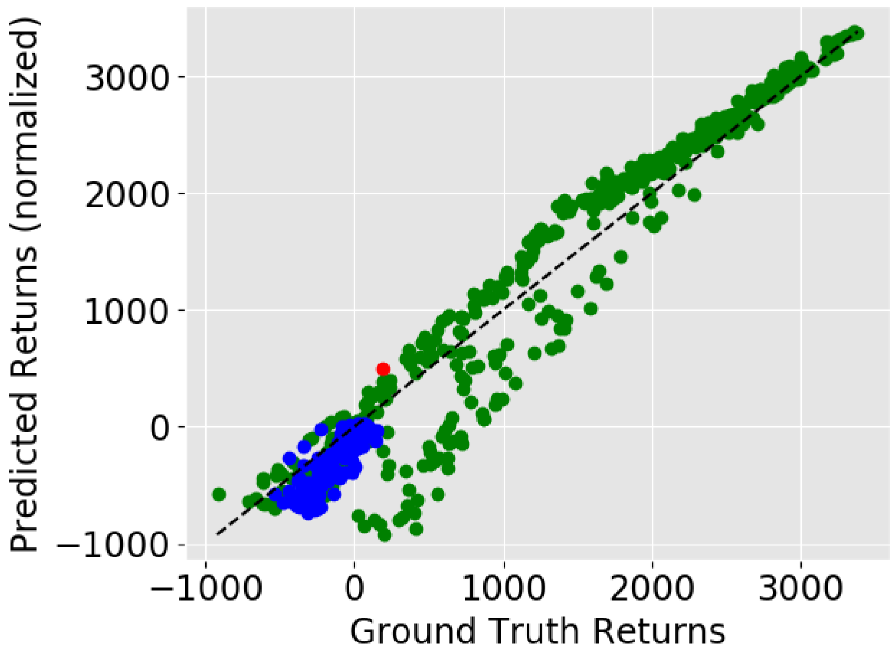
We identified that both algorithms' performances are highly related to the generated noisy trajectories, i.e., different random seeds generate different noisy trajectories and it could dramatically influence the following reward learning process. Also, D-REX uses its

Behavior Cloning policy in trajectory generation while SSRR utilizes AIRL's policy. In order to make an apple-to-apple comparison, we have to fix the same noisy trajectory dataset for the two algorithms to learn and to compare. Therefore, we generated five noisy datasets from Behavior Cloning and five noisy datasets from AIRL and criticizes all datasets through AIRL's reward function for SSRR.

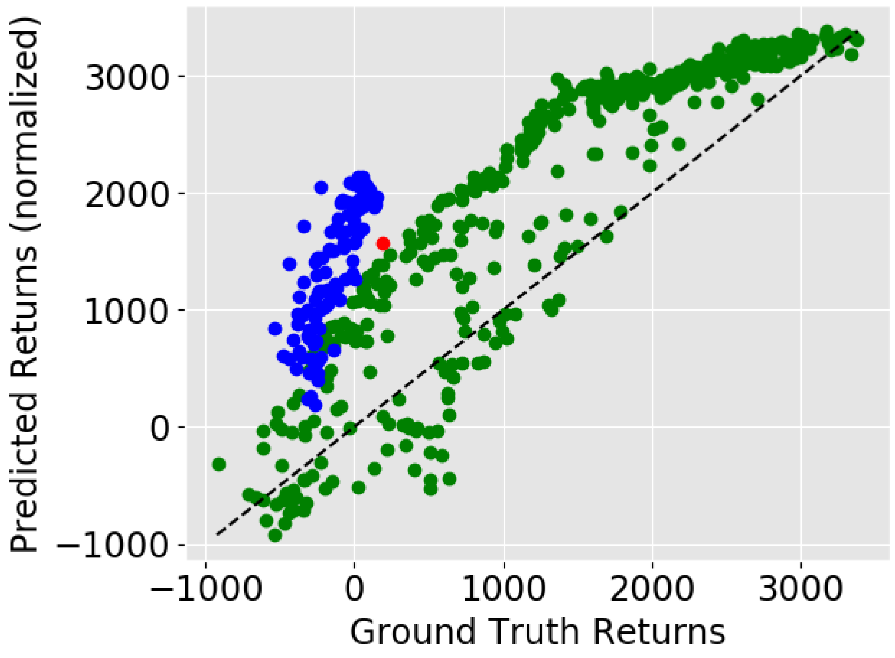
The result of running SSRR and D-REX on Behavior Cloning dataset is shown in Table 3.1. Since SSRR and D-REX learn from the same dataset for each row, we could apply paired *t*-test to examine the difference. According to Shapiro-Wilk test for normality, we cannot reject the Gaussian null hypothesis for the paired differences between SSRR and D-REX ($p = 0.965$). Further, single-tail paired *t*-test shows SSRR's correlation is not significantly higher than D-REX ($t = 1.208, p = 0.147$).

For AIRL dataset, the result of correlation coefficient between SSRR and D-REX is shown in Table 3.2. Similarly, we cannot reject the Gaussian null hypothesis for the paired differences ($p = 0.703$). Single-tail paired *t*-test shows SSRR's correlation is significantly higher than D-REX ($t = 8.358, p = 0.0005$). As such, we argue the reason why SSRR with AIRL data works much better than SSRR with BC data is that AIRL's initial reward $\tilde{R}$ has better knowledge to criticize its paired policy, $\tilde{\pi}$. In contrast, BC trajectory could deviate from what the initial reward $\tilde{R}$ learns during training. This highlights the need to get a reasonably reliable initial reward $\tilde{R}$ that could criticize the corresponding noisy trajectories, which we will have further discussion in Section 3.5.

We also show reward correlations between learned reward and ground-truth reward of SSRR and D-REX from one example run in Figure 3.5a and 3.5b. Red dots represent the demonstration given to BC/AIRL to learn from. Blue dots are the noise injected trajectories, which are also D-REX and SSRR's direct learning source. Green dots are unseen trajectories that are used to "test" the learned reward function. We note that the same set of "green dots" (test set) is used across this Chapter. Both SSRR returns and D-REX returns are normalized to be in the same magnitude as ground-truth returns, which will not



(a) Correlation with Ground-truth reward from our method, SSRR, on HalfCheetah.



(b) Correlation with Ground-truth reward from D-REX on HalfCheetah.

Figure 3.5: Learned Reward Function's Correlation with Ground-Truth Reward, Compared between Our Method, SSRR and D-REX, on HalfCheetah.

affect the meaning of the reward function, as reward functions are invariant under affine transformation.

3.4.2 Hopper

Demonstrations we use for Hopper experiment are 400 demonstrations with their averaged undiscounted cumulative rewards of 1029.1, while optimal policy typically has more than 2000 rewards.

The Hopper environment is more tricky because, unlike HalfCheetah, the default Hopper environment will terminate the episode when the agent falls down. Further, the agent cannot rely on its three joints to stably "stand" in the environment. Combining these two facts, a trivial reward function of always give $+1$ reward for each timestep alive in the environment could easily help the RL algorithm to learn the optimal behavior, since once the agent stops hopping forward, it dies and stops receiving the positive reward. We show the correlation between the lengths of episodes (equivalent to $+1$ reward for each timestep) and ground-truth returns in Figure 3.6, which has a correlation coefficients of 0.997. Thus, in order to avoid the trivial solution, we keep the agent alive even when it falls, and it could still crawl forward, as shown in Figure 2.5. Unfortunately, after such change to the environment, both SSRR and D-REX fail to learn a reasonable reward function. One drawback of SSRR with the AIRL noisy dataset is that AIRL did not learn a good-enough initial reward so that SSRR could not find an accurate sigmoid function. This weakness motivates our next section, which aims to model suboptimal demonstrations directly in the reward function even in the noisy data generation process.

3.5 Optimality-Parameterized Adversarial Inverse Reinforcement Learning

The main problem in the previous SSRR approach is that the its success depends on reliable initial reward learning, especially the accuracy on the noisy trajectories that SSRR will learn from. For the case of suboptimal demonstration, AIRL might fail to learn an

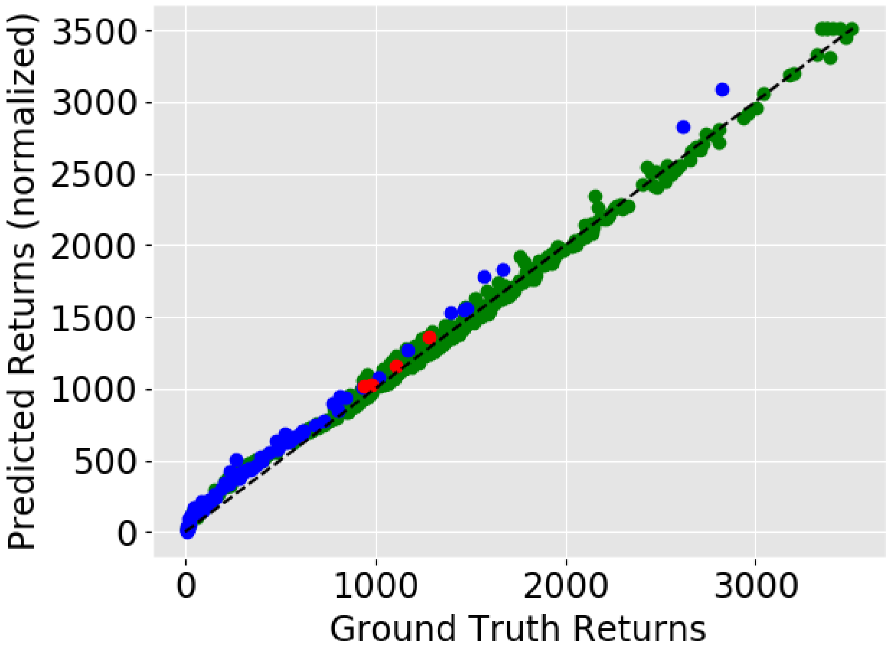


Figure 3.6: Episode Length Correlation with Ground-Truth Reward in Hopper

accurate model of the idealized reward because it assumes that the demonstrations - in this case, suboptimal demonstrations - are the idealized demonstrations. Besides, all the generated state-action pairs during AIRL training are noise-free, and thus $\tilde{R}$ might not have the greatest ability to criticize noisy trajectories, which is exactly what we use for SSRR learning.

In light of that limitation, we propose to modify AIRL's discriminator definition to let it learn from demonstrations with different noise levels, which could help carve the reward function better. We call the method Optimality-Parameterized Adversarial Inverse Reinforcement Learning (OP-AIRL). It is worth noting that OP-AIRL itself should have the ability to model a better reward than original AIRL, making it possible to have its policy better than the best demonstration. We could also combine OP-AIRL with SSRR to achieve even better performance than stand-alone OP-AIRL or SSRR.

OP-AIRL has three major differences from AIRL, illustrated in Figure 3.7. First, the discriminator takes an extra input, which is the known noise level of the demonstration, and

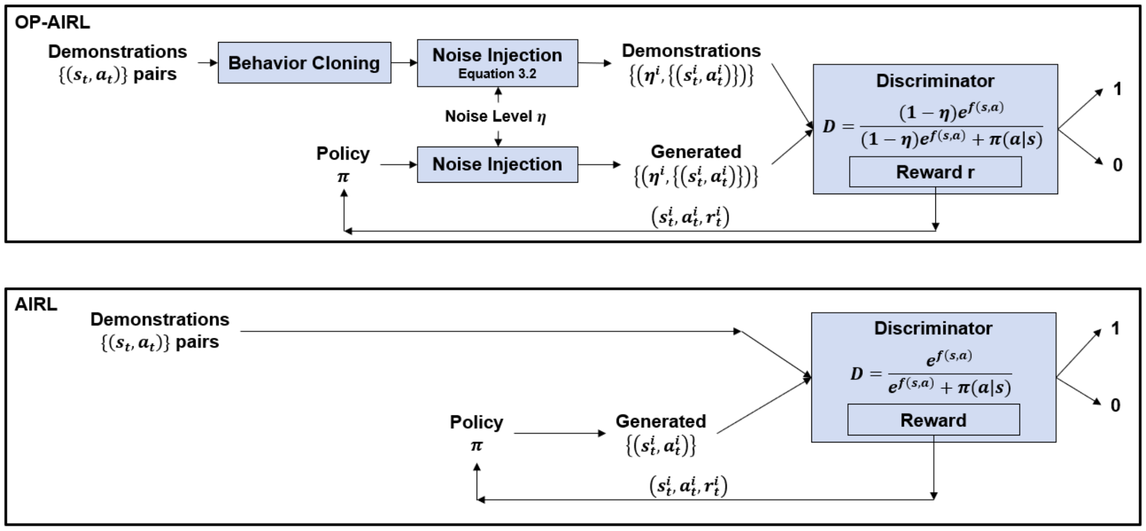


Figure 3.7: Optimality-Parameterized Adversarial Inverse Reinforcement Learning Diagram.

the discriminator is changed to

$$D = \frac{(1-\eta)e^{f(s,a)}}{(1-\eta)e^{f(s,a)} + \pi(a|s)}. \tag{3.9}$$

The intuition behind the change is that when $\eta = 0$, the objective is as usual. When $\eta = 1$, since the result of $e^{f(s,a)}$ is multiplied with 0, the gradient will stop and essentially the reward function will not be updated, which should be the case since $\eta = 1$ means totally-random policy. Thus, $1 - \eta$ works similar to a confidence weight for the incoming state-action pairs (the higher state-action's $1-\eta$ is, the more the reward will learn from the state-action pair), whether demonstrated or generated. The second change is the demonstration now needs to be noisy trajectories. To accomplish such goal, we simply apply a Behavior Cloning + Noise Injection to obtain noisy trajectories just as D-REX. The third change is the generated trajectory also needs to be noisy. Thus, we inject noise η when generating policy's trajectories as well.

After the three changes, OP-AIRL now takes and generates noisy trajectories and learns the reward function with known noise level information.

3.6 Results of OP-AIRL

In this section, we first present the stand-alone OP-AIRL results in terms of its ability to learn a policy of higher quality than the original demonstrations. Then we show the results to utilize OP-AIRL as a method to generate better training data for SSRR. With better data for SSRR, we can then more efficiently train a policy to transcend the quality of the initial demonstration.

3.6.1 OP-AIRL

We apply OP-AIRL to both HalfCheetah and Hopper, and both achieved a policy better than best demonstration.

On HalfCheetah, we again use the single 187.7 undiscounted cumulative reward demonstration, and OP-AIRL's policy is able to achieve 591 (316%)[1] undiscounted cumulative reward in its best trajectory. The average return of the final 100 episodes of OP-AIRL is 270 (144%), which is higher than average return of the final 100 episodes of AIRL, 150. Note that AIRL's final results are below the demonstration while OP-AIRL's is higher than the demonstration. The learning curve of OP-AIRL and AIRL is shown in Figure 3.8. Not only OP-AIRL could achieve better-than-best-demonstration performance, it is also sample efficient compared with original AIRL.

On Hopper, we use 400 demonstrations with a mean performance of 1029.1. The resulting OP-AIRL policy achieves 2521 (245%) average return of the final 100 episodes and a maximum of 3200 (311%) return, shown in Figure 3.9. In contrast, AIRL only achieves 1083 average return of the final 100 episodes and a maximum of 1680 return.

[1]The number in parenthesis represents the percent improvement over the ground truth reward of the original demonstration.

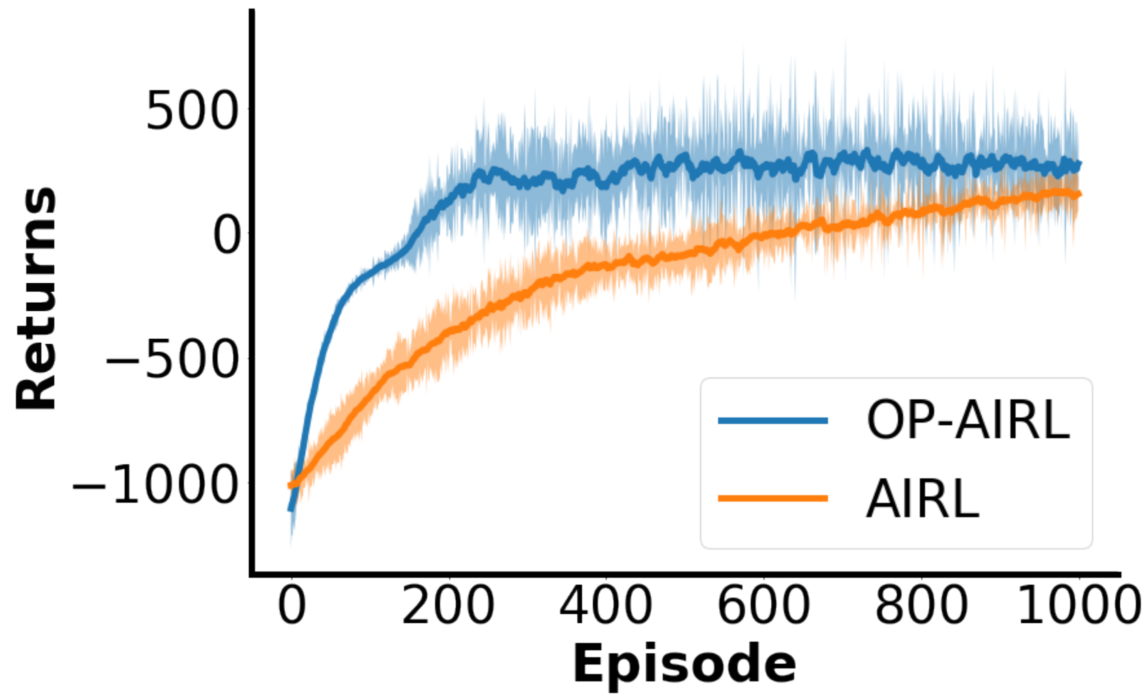


Figure 3.8: OP-AIRL and AIRL Learning Curve on HalfCheetah. Solid lines are mean of 3 repeats. Shadow represents standard deviation across 3 repeats. Each data point is average of 10 trajectories generated in one episode.

3.6.2 OP-AIRL+SSRR

The idea of combining OP-AIRL and SSRR is straightforward: substitute the data generation block of SSRR (for reference, see Figure 3.4) with OP-AIRL (see Figure 3.7). In this manner, we could get a relatively reliable initial reward $\tilde{R}$ to fit an accurate sigmoid performance-noise relationship, and then learn reward and policy based on the more-accurate reward signal.

HalfCheetah

OP-AIRL+SSRR's reward correlation with ground-truth reward on HalfCheetah is shown in Table 3.3. For the conveinence of comparison, Table 3.3 also includes results in Table 3.1 and Table 3.2. Note that for each row, the correlation coefficients are not paired comparison since their noisy dataset is different (coming from different noisy dataset generation methods). Nonetheless, the same OP-AIRL generated noisy dataset is fixed for

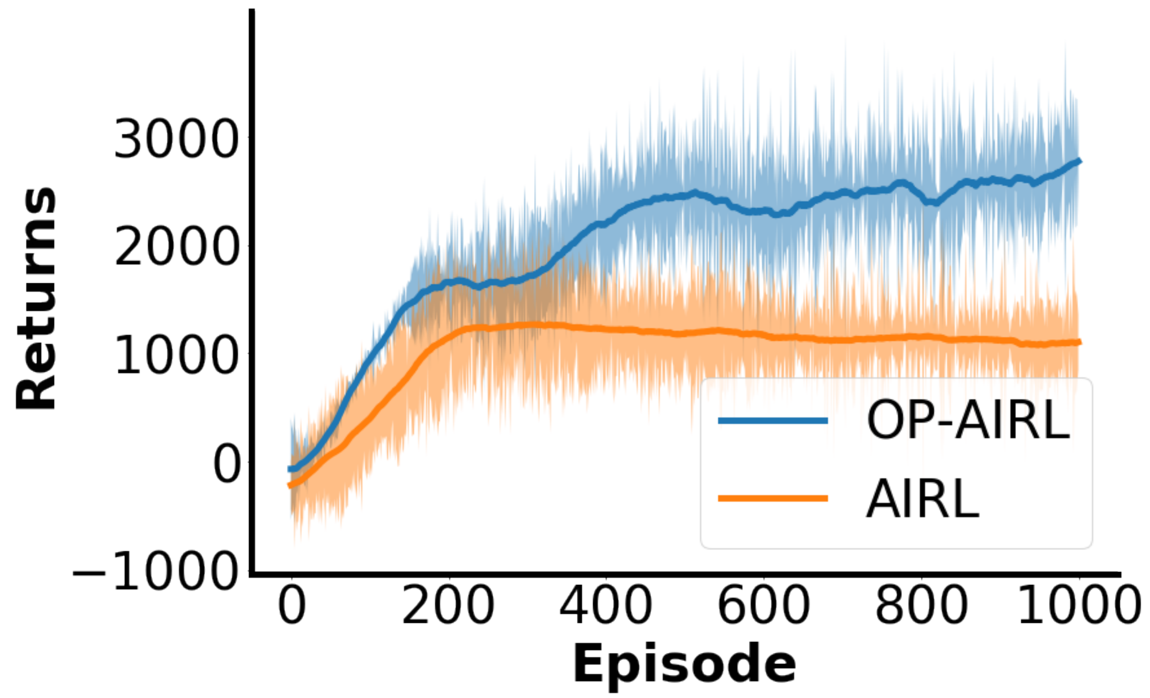


Figure 3.9: OP-AIRL and AIRL Learning Curve on Hopper. Solid lines are mean of 3 repeats. Shadow represents standard deviation across 3 repeats. Each data point is average of 10 trajectories generated in one episode.

Table 3.3: Learned Reward Correlation Coefficients with Ground-Truth Reward Comparison between SSRR and D-REX with different noisy trajectory generation methods on HalfCheetah

Repeat	BC		AIRL		OP-AIRL	
	D-REX	SSRR	D-REX	SSRR	D-REX	SSRR
1	0.902	0.917	0.830	0.911	0.668	0.958
2	0.859	0.936	0.804	0.962	0.780	0.942
3	0.917	0.889	0.810	0.943	0.614	0.892
4	0.933	0.933	0.832	0.935	0.762	0.988
5	0.866	0.912	0.814	0.912	0.719	0.964
Mean (Std. Dev.)	0.900 (0.025)	0.929 (0.017)	0.818 (0.011)	0.933 (0.019)	0.709 (0.061)	0.948 (0.032)

OP-AIRL+D-REX and OP-AIRL+SSRR to provide apple-to-apple comparison between SSRR and D-REX. An example of the correlation of the recovered reward functions of SSRR and D-REX are shown in Figure 3.10a and 3.10b, respectively. Similarly, red dots

Table 3.4: Learned Reward Correlation Coefficients with Ground-Truth Reward Comparison between OP-AIRL+SSRR and OP-AIRL+D-REX on Hopper

Repeat	OP-AIRL + D-REX	OP-AIRL + SSRR
1	0.048	0.919
2	0.159	0.902
3	-0.076	0.832
4	-0.020	0.950
5	0.011	0.929
Mean (Std. Dev.)	0.024 (0.079)	0.906 (0.040)

represent the demonstration given to BC or AIRL to learn from; blue dots are the noise injected trajectories output by OP-AIRL; and green dots are unseen trajectories that are used to “test” the learned reward function.

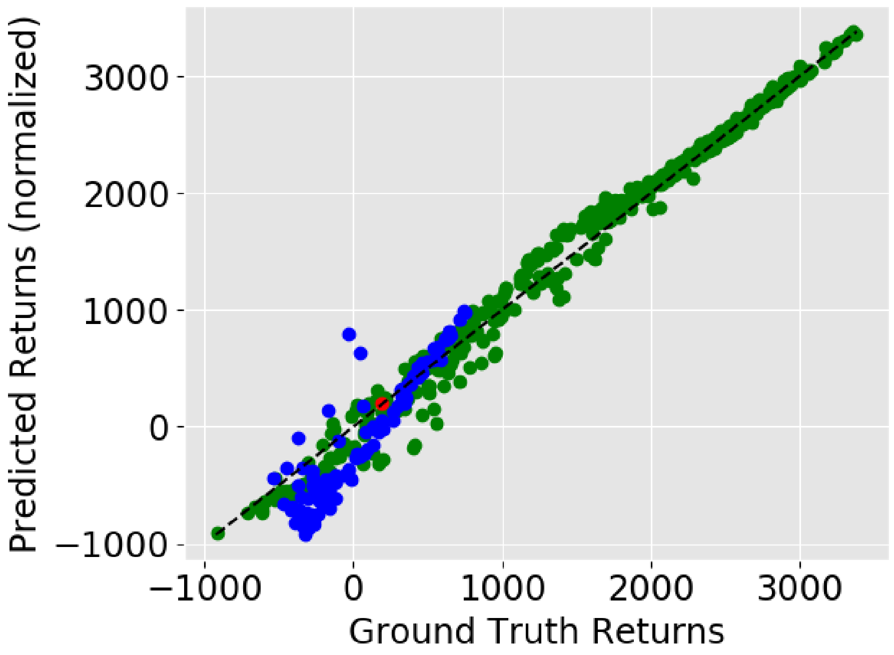
We further train reinforcement learning on the reward from SSRR; the results are shown in Figure 3.11. We are able to achieve mean performance of 1286 (688%) on three repeats with a standard deviation of 217, which means we surpass the original demonstration (187 performance). In comparison, D-REX only achieves 816 performance as the best one out of three repeats[2].

Hopper

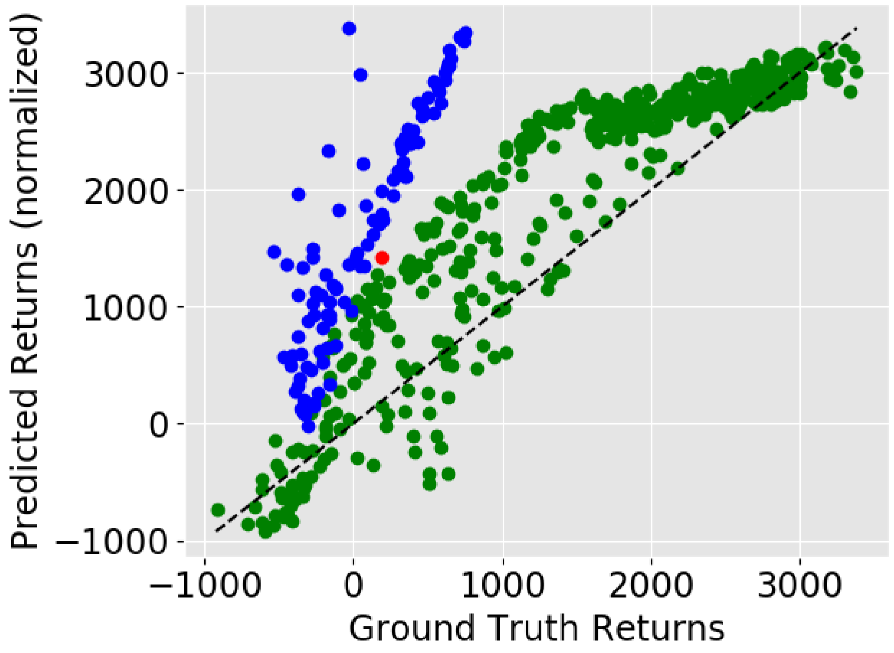
With OP-AIRL, SSRR could successfully recover highly correlated reward functions while D-REX still completely fails, as shown in Table 3.4.

We also train reinforcement learning on the reward from SSRR on Hopper, which results in a mean performance of 2611 (254%) on three repeats with 517 standard deviation, surpassing the original demonstration (1029 performance). Note that on a default Hopper environment, any positive reward each timestep could just lead to an optimal policy (see Section 3.4 for detail), but here we consider the “un-dying” Hopper environment. Thus,

[2]We were not able to reproduce the original performance that D-REX paper reported (972 with std 96.1). We believe this lack of replication is due to large variance of final results on different noisy datasets, as we mentioned in Section 3.4. Nonetheless, our new results are significantly higher than the reported results, too.



(a) Learned Reward Correlation with Ground-Truth Reward of OP-AIRL + SSRR on HalfCheetah



(b) Learned Reward Correlation with Ground-Truth Reward of OP-AIRL + D-REX on HalfCheetah

Figure 3.10: Learned Reward Function's Correlation with Ground-Truth Reward, Compared between Our Method, OP-AIRL + SSRR and OP-AIRL + D-REX, on HalfCheetah.

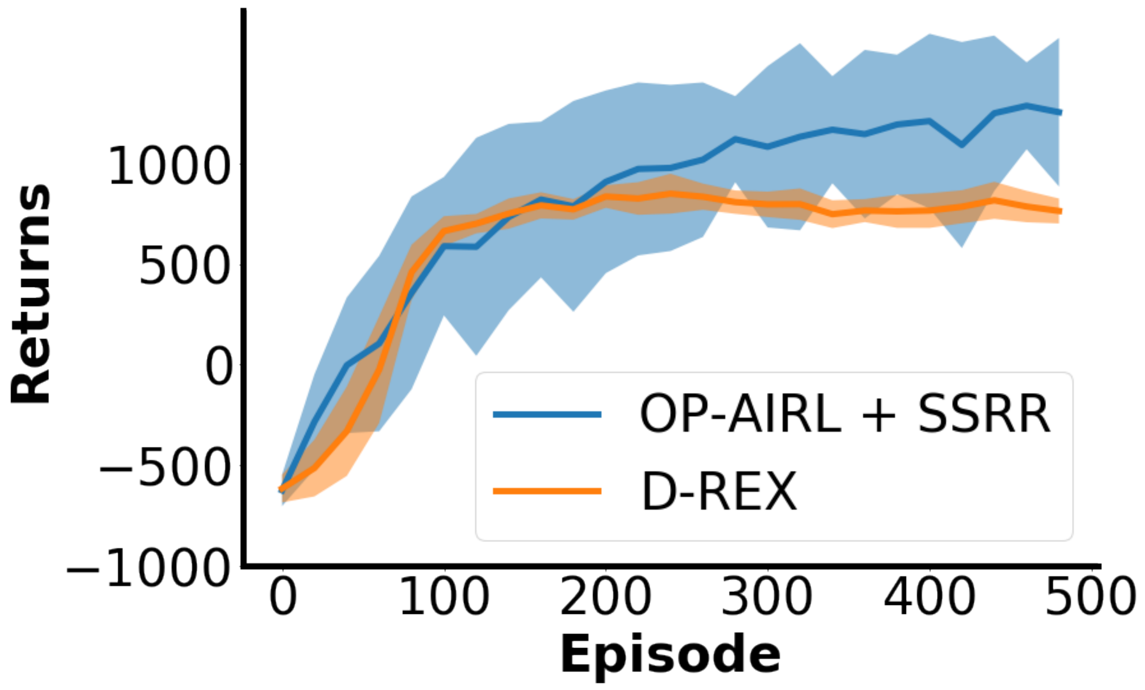


Figure 3.11: Learning Curve of Reinforcement Learning on Reward Learned by D-REX and OP-AIRL + SSRR

comparing it with D-REX paper's reported 2072 performance is not a true test of the algorithm's performance. As such, we focus on our modified Hopper environment where one cannot learn a reward function by simply correlating the length of the trajectory with the reward itself.

3.7 Discussion and Future Work

3.7.1 Discussion

In this chapter, we aim to learn from heterogeneous-performance demonstrations, especially the case when all demonstrations are suboptimal. Despite not having access to ranked demonstrations from humans to then regress a reward function directly, our approach nonetheless is able to intelligently bootstrap off of a single, suboptimal demonstration to learn a reward function of a latent, more-optimal demonstration. Further, our approach is able to adapt its bootstrapping approach to each domain, which is an important improvement over prior literature (D-REX [52]) that assumes a specific relationship between task performance and a noise-injection model that is task agnostic. In order to have a better mechanism for generating helpful training data, we further introduce OP-AIRL. Combining OP-AIRL and SSRR, we obtain much higher performance policy than the best demonstration available.

3.7.2 Future Work

In future work, we aim to apply OP-AIRL with SSRR on a physical robotics platform as we have done for MSRD (Chapter 2). One of the considerations we must have when applying on physical robots is the safety concern, especially when learning from suboptimal demonstrations, as the robot might try to explore wildly.

It would also be valuable to provide more theoretical support for OP-AIRL and SSRR. For example, is sigmoid function powerful enough to capture the performance-noise relationship? Can we bound the error that initial reward has to make sure the success of sigmoid fitting?

Another interesting related topic is heterogeneous-performance where there is mix of near-optimal demonstrations and suboptimal demonstrations. Is it possible to identify both classes? Will the suboptimal demonstrations still be useful even when near-optimal demonstration is available, like semi-supervised learning?

CHAPTER 4
CONCLUSION AND FUTURE WORK

4.1 Conclusion

In this thesis, I present contributions for my research that advance the capability of Learning from Demonstration techniques to heterogeneous strategy scenarios and heterogeneous performance cases.

I began in Chpater 1 with an introduction to the problem of interest for this thesis.

In Chapter 2, I present an algorithm, Multi-Strategy Reward Distillation (MSRD), to tackle the learning from heterogeneous strategy demonstration problem. This method separates the task reward component from strategy reward components to achieve both better task reward recovery and better strategy reward estimation. It is proved to be effective on two simulated robotic tasks and one real-world robot task.

Next, in Chpater 3, I developed two novel algorithms, Self-Supervised Reward Regression (SSRR) and Optimality-Parameterized Adversarial Inverse Reinforcement Learning (OP-AIRL), to deal with the learning from heterogeneous performance (suboptimal) demonstrations. Combining the both algorithms provides us with the ability to not only learn from heterogeneous performance demonstrations but also a much better final policy than the best demonstrations available. I showed the effectiveness of OP-AIRL with SSRR on two simulated robotic tasks.

4.2 Future Work

For learning from heterogeneous strategy demonstration, a direct extension would be to combine it with strategy-inference methods or joint inference techniques to remove MSRD's dependence on strategy labelling. Moreover, I also expect to introduce an empirical covariance minimization loss between different learned reward functions, which could further de-entangle different reward components.

For learning from heterogeneous performance demonstration, reducing the system's complexity might be an interesting topic, which could help reduce the variance of the approach and decrease the difficulty to tune the algorithm to new environments. Human-in-the-loop approaches could also be considered to reduce the tuning effort in the heterogeneous performance LfD problem, as it is not too expensive to query human about ranking of trajectories. Combining near-optimal demonstration with suboptimal demonstration is also a promising direction.

For other types of heterogeneity, such as heterogeneous environments, heterogeneous embodiments and heterogeneous tasks, it is possible to utilize meta-learning and Cycle-GAN to fulfill the transfer between environments and embodiments, and employ sub-task (skill) learning to accomplish shared learning between tasks.

REFERENCES

[1] M. Cakmak and L. Takayama, “Towards a comprehensive chore list for domestic robots,” in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, IEEE, 2013, pp. 93–94.

[2] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schaeffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, “The kuka-dlr lightweight robot arm - a new reference platform for robotics research and manufacturing,” in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, 2010, pp. 1–8.

[3] R. R. Murphy, *Disaster Robotics*. The MIT Press, 2014, ISBN: 0262027356.

[4] A. Sparkes, W. Aubrey, E. Byrne, A. Clare, M. N. Khan, M. Liakata, M. Markham, J. Rowland, L. N. Soldatova, K. E. Whelan, *et al.*, “Towards robot scientists for autonomous scientific discovery,” *Automated Experimentation*, vol. 2, no. 1, p. 1, 2010.

[5] 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, 2020.

[6] S. Chernova and A. L. Thomaz, “Robot learning from human teachers,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.

[7] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] P. Domingos, “A few useful things to know about machine learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[9] H. A. Simon, “Theories of bounded rationality,” *Decision and organization*, vol. 1, no. 1, pp. 161–176, 1972.

[10] S. Nikolaidis, S. Nath, A. D. Procaccia, and S. Srinivasa, “Game-theoretic modeling of human adaptation in human-robot collaboration,” in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI ’17, Vienna, Austria: ACM, 2017, pp. 323–331, ISBN: 978-1-4503-4336-7.

[11] A. Newell, H. A. Simon, *et al.*, *Human problem solving*, 9. Prentice-Hall Englewood Cliffs, NJ, 1972, vol. 104.

[12] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Imitation in animals and artifacts," K. Dautenhahn and C. L. Nehaniv, Eds., pp. 171–189, 2002.

[13] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova, "Robot web tools: Efficient messaging for cloud robotics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 4530–4537.

[14] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," AAI3438449, PhD thesis, Pittsburgh, PA, USA, 2010, ISBN: 978-1-124-41421-8.

[15] L. Smith, N. Dhawan, M. Zhang, P. Abbeel, and S. Levine, "Avid: Learning multi-stage tasks via pixel-level translation of human videos," *arXiv preprint arXiv:1912.04443*, 2019.

[16] *Leetcode*, `https://leetcode.com/`, Accessed: 2020-04-22.

[17] L. Chen, R. R. Paleja, M. Ghuy, and M. C. Gombolay, "Joint goal and strategy inference across heterogeneous demonstrators via reward network distillation," in *HRI '20: ACM/IEEE International Conference on Human-Robot Interaction, Cambridge, United Kingdom, March 23-26, 2020*, T. Belpaeme, J. Young, H. Gunes, and L. D. Riek, Eds., ACM, 2020, pp. 659–668.

[18] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, 2011, pp. 627–635.

[19] N. D. Daw and P. Dayan, "The algorithmic anatomy of model-based evaluation," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1655, p. 20 130 478, 2014.

[20] G. Klein, "The recognition-primed decision (rpd) model: Looking back, looking forward," *Naturalistic decision making*, pp. 285–292, 1997.

[21] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, 2000, p. 2.

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

[23] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pp. 729–736.

[24] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning.," in *IJCAI*, vol. 7, 2007, pp. 2586–2591.

[25] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, ser. AAAI'08, Chicago, Illinois: AAAI Press, 2008, pp. 1433–1438, ISBN: 978-1-57735-368-3.

[26] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[27] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning*, 2016, pp. 49–58.

[28] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *International Conference on Learning Representations*, 2018.

[29] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 4565–4573.

[30] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim, "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 1235–1245.

[31] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 3812–3822.

[32] M. Gombolay, X. J. Yang, B. Hayes, N. Seo, Z. Liu, S. Wadhwania, T. Yu, N. Shah, T. Golen, and J. Shah, "Robotic assistance in the coordination of patient care," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1300–1316, 2018.

[33] M. Gombolay, R. Jensen, J. Stigile, S.-H. Son, and J. Shah, "Apprenticeship scheduling: Learning to schedule from human experts," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, AAAI Press, 2016, pp. 826–833.

[34] M. Gombolay, R. Jensen, J. Stigile, T. Golen, N. Shah, S.-H. Son, and J. Shah, "Human-machine collaborative optimization via apprenticeship scheduling," *Journal of Artificial Intelligence Research*, vol. 63, pp. 1–49, 2018.

[35] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*, Springer, 2007, pp. 291–324.

[36] K. Xu, E. Ratner, A. Dragan, S. Levine, and C. Finn, "Learning a prior over intent via meta-inverse reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, Long Beach, California, USA: PMLR, 2019, pp. 6952–6962.

[37] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *Proceedings of the 9th European Conference on Recent Advances in Reinforcement Learning*, ser. EWRL'11, Athens, Greece: Springer-Verlag, 2012, pp. 273–284, ISBN: 978-3-642-29945-2.

[38] J. Choi and K. eung Kim, "Nonparametric bayesian inverse reinforcement learning for multiple reward functions," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 305–313.

[39] K. Amin, N. Jiang, and S. Singh, "Repeated inverse reinforcement learning," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 1815–1824.

[40] B. Woodworth, F. Ferrari, T. E. Zosa, and L. D. Riek, "Preference learning in assistive robotics: Observational repeated inverse reinforcement learning," in *Proceedings of the 3rd Machine Learning for Healthcare Conference*, F. Doshi-Velez, J. Fackler, K. Jung, D. Kale, R. Ranganath, B. Wallace, and J. Wiens, Eds., ser. Proceedings of Machine Learning Research, vol. 85, Palo Alto, California: PMLR, 2018, pp. 420–439.

[41] S. Nikolaidis, K. Gu, R. Ramakrishnan, and J. A. Shah, "Efficient model learning for human-robot collaborative tasks," *CoRR*, vol. abs/1405.6341, 2014. arXiv: `1405.6341`.

[42] P. Henderson, W.-D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup, "Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[43] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[44] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[45] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4496–4506.

[46] W. M. Czarnecki, R. Pascanu, S. Osindero, S. Jayakumar, G. Swirszcz, and M. Jaderberg, "Distilling policy distillation," in *Proceedings of Machine Learning Research*, K. Chaudhuri and M. Sugiyama, Eds., ser. Proceedings of Machine Learning Research, vol. 89, PMLR, 2019, pp. 1331–1340.

[47] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. eprint: `arXiv:1606.01540`.

[48] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033.

[49] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *International Conference on Learning Representations*, 2019.

[50] Y. Zhang, W. Yu, and G. Turk, "Learning novel policies for tasks," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, Long Beach, California, USA: PMLR, 2019, pp. 7483–7492.

[51] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[52] D. S. Brown, W. Goo, and S. Niekum, "Ranking-based reward extrapolation without rankings," *CoRR*, vol. abs/1907.03976, 2019. arXiv: `1907.03976`.

[53] S. K. S. Ghasemipour, R. Zemel, and S. Gu, “A divergence minimization perspective on imitation learning methods,” *arXiv preprint arXiv:1911.02256*, 2019.

[54] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum, “Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations,” *arXiv preprint arXiv:1904.06387*, 2019.