The Dissertation Committee for Daniel Sundquist Brown
certifies that this is the approved version of the following dissertation:

Safe and Efficient Inverse Reinforcement Learning

Committee:

Scott Niekum, Supervisor

Peter Stone

Ufuk Topcu

Anca Dragan

Safe and Efficient Inverse Reinforcement Learning

by

Daniel Sundquist Brown

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2020

For Baby Boopums

Thank you for not coming until after my defense.

Acknowledgments

Completing this dissertation would not have been possible without the help and support from many people. I would especially like to thank my advisor Scott Niekum for giving me the flexibility and freedom to pursue my interests and providing much needed support, useful advice, and quality mentoring along the way. Thank you Scott for helping me see the big picture and helping me maintain a positive outlook on my research throughout my PhD. A special thanks also goes to my dissertation committee members, Peter Stone, Ufuk Topcu, and Anca Dragan, for their great feedback and advice.

During graduate school I had the pleasure of getting to know and working with many other students at UT. I would especially like to thank the members of the Personal Autonomous Robotics Lab and the other graduate students at UT for their friendship, support, collaboration, and many great discussions. A special thanks goes to Caleb Chuck, Russell Coleman, Yuchen Cui, Ishan Durugkar, Taylor Kessler Faulkner, Wonjoon Goo, Prasoon Goyal, Alex Gutierrez, Josiah Hanna, Ajinkya Jain, Rudi Lioutikov, Akanksha Saran, Jordan Schneider, and Faraz Torabi. I have also had great collaborations outside of UT and I would like to especially thank Prabhat Nagarajan and Marek Petrik.

Before beginning my PhD I was fortunate to work as a research scientist at the Air Force Research Lab's Information Directorate. I learned much during my time at AFRL and I greatly appreciate the valuable research experiences I had that prepared me for completing my PhD. Thank you Matt Berger, Nate Gemelli, Oliver Hennigh, Jeff Hudack, Ainoghena Igetei, Steve Loscalzo, Lee Seversky, Ryan Turner, and Bob Wright for your camaraderie

and mentorship. Before working at AFRL I had the pleasure of attending Brigham Young University and completing a master's degree with Mike Goodrich and an undergraduate thesis with Sean Warnick. Thank you Sean for introducing me to research and the interesting world of financial risk analysis. Thank you Mike for introducing me to human-robot interaction and for your invaluable mentorship that helped me gain confidence as a young researcher. While I couldn't have guessed it at the time, my research experiences as an undergraduate and graduate student were both beneficial to completing my PhD: much of my dissertation applies financial risk metrics to the study of imitation learning, a core problem in human-robot interaction.

Finally, the real reason I was able to finish this dissertation is because of my wonderful family. Thank you Mom and Dad for always providing support and encouraging me throughout my education. Thank you Ryan, Jared, Nicole, Michelle, Carson, Spencer, Lauren, Kylie, and Brycen for being the best siblings ever. Thank you Sarah for your loving support and encouragement and the many sacrifices you made to help me during my PhD. Thank you William and Emma for the fun play dates we had in Austin that gave me much needed breaks from research as well as time to step away from my computer and think about my research more broadly and creatively.

DANIEL SUNDQUIST BROWN

The University of Texas at Austin

August 2020

Safe and Efficient Inverse Reinforcement Learning

by

Daniel Sundquist Brown, Ph.D.
The University of Texas at Austin, 2020

Supervisor: Scott Niekum

As robots and other autonomous agents enter our homes, hospitals, schools, and workplaces, it is important that they can safely and efficiently infer and adapt to human preferences. One common way to teach human preferences to robots and other autonomous agents is through imitation learning, where an agent learns by observing demonstrations of how to perform a task. Imitation learning has the potential to allow everyday users the ability to program and adapt the behavior of an autonomous agent simply by showing it how to perform a task. However, for imitation learning algorithms to be deployed in complex, possibly high-risk situations, it is important that these algorithms can provide practical, high-confidence bounds on performance.

If a robot is to reason effectively about its performance when learning from demonstrations, it needs to infer the goals and intent of the demonstrator. One common way to

infer goals and intentions from demonstrations is through inverse reinforcement learning, where the goal is to infer the reward function of the demonstrator. However, most inverse reinforcement learning algorithms have limited real-world applicability because they do not provide practical assessments of safety, often require large numbers of demonstrations, and have high computational costs. This dissertation addresses these shortcomings by developing efficient inverse reinforcement learning algorithms that allow autonomous agents to provide high-confidence bounds on performance when learning from demonstrations.

We first formalize the problem of safe imitation learning via high-confidence performance bounds. We then present a general Bayesian framework for computing tight high-confidence performance bounds on any evaluation policy when the true reward function is unknown and must be inferred from demonstrations. The method we propose is sample-efficient, but is computationally inefficient for learning in complex, high-dimensional tasks. To address this computational inefficiency, we first introduce a computationally efficient algorithm for reward learning via ranked demonstrations. We show that preference rankings over demonstrations enable reward inference algorithms to scale to high-dimensional imitation learning tasks such as learning to play Atari games with no access to the score, but with access to a few suboptimal, ranked demonstrations. We also show that preference rankings allow for better-than-demonstrator performance and that rankings over demonstrations can sometimes be obtained automatically, without requiring access to explicit preference labels. Furthermore, we leverage the computational efficiency of reward learning via preferences to scale high-confidence policy evaluation to complex imitation learning settings with high-dimensional, visual demonstrations.

While our work on high-confidence policy evaluation gives efficient bounds on the performance of an imitation learning agent, it does not answer the question of what an agent should do to learn a policy that is safe with high probability. The final contributions of this dissertation are two different approaches for robust policy optimization for imitation learning. We first derive an algorithm that directly optimizes a policy to balance risk and

expected return under a reward function posterior given a fixed set of demonstrations. Second, we address the problem of robust policy optimization via active learning. We present a sample-efficient, active inverse reinforcement learning algorithm that generates risk-aware queries that enable robust policy improvement via repeated interactions with a demonstrator.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As robots and other autonomous agents enter our homes, hospitals, schools, and workplaces, it is important that they can safely and efficiently infer and adapt to human preferences. In particular, it is critical that robots can be customized by anyone, especially people who are not robotics engineers. One common way to teach preferences to robots and other autonomous agents is through imitation learning, where an agent learns by observing demonstrations of how to perform a task. Imitation learning is important because it provides both experts and non-experts an intuitive way to program and adapt intelligent systems by simply demonstrating how they would like a task to be accomplished, without requiring them to hard code behaviors or write down an explicit reward function for the system to optimize via reinforcement learning. Furthermore, even if a reward function is available, it can be difficult to learn a good control policy without the guidance of a few demonstrations (Zhu et al., 2018; Thananjeyan et al., 2019).

Much work on imitation learning seeks to directly find a policy from demonstrations via behavioral cloning, where the goal is to learn to mimic the demonstrator (Pomerleau, 1991; Torabi et al., 2018a; Ijspeert et al., 2013; Paraschos et al., 2013); however, while these methods teach a robot *how* to act like the demonstrator, they do not allow explicit reasoning about *why* the demonstrator acted in certain ways. Without an understanding of why the

demonstrator performed certain actions or visited certain states, an imitation learning policy may fail to generalize to new situations, which can lead to dangerous behavior in high-risk domains. Thus, an open and important challenge in imitation learning is to determine the safety and robustness of a learned policy with respect to the demonstrator's true, but unknown intent. One common way to perform intent inference is through inverse reinforcement learning, where the goal is to infer a reward function that models the demonstrator's intent (Ng and Russell, 2000; Abbeel and Ng, 2004). Recovering the intent of the demonstrator by learning a reward function allows a learning agent to explain the behavior of the demonstrator and also to optimize its own behavior using reinforcement learning (Sutton and Barto, 1998), in order to successfully imitate the demonstrator. Recovering a reward function also provides a way to potentially generalize to novel tasks or embodiments (Fu et al., 2017), allows a learner to potentially improve upon the performance of the demonstrator (Brown et al., 2019b,a), and provides a way to explicity measure costs and reason about safety (Garcıa and Fernández, 2015; Brown and Niekum, 2018; Brown et al., 2018, 2020). However, despite the advantages of imitation learning via inverse reinforcement learning, current inverse reinforcement learning algorithms have limited real-world applicability because they (1) do not provide practical assessments of safety, (2) often require large numbers of demonstrations, and (3) have high computational costs. This dissertation will address each of these limitations.

The main focus of this dissertation is achieving safe imitation learning via inverse reinforcement learning (IRL). In this dissertation we define a safe algorithm as one that provides a measure of confidence in the correctness of the output of the algorithm. Rather than giving many demonstrations to an imitation learning agent and then hoping that it learns a good policy, we want imitation learning algorithms that provide high-confidence bounds on performance. However, despite the importance and increased interest in safety in machine learning and robotics (Wyrobek et al., 2008; Amodei et al., 2016; Thomas et al., 2019; Fisac et al., 2018), safe imitation learning is currently a hard, open problem that is not easily

achieved. Indeed, even if a robot could perfectly intuit a demonstrator's reward function, it would still face the difficult problem of safe policy optimization via reinforcement learning (Garcıa and Fernández, 2015).

This dissertation addresses the problem of safe imitation learning by answering the following question:

How can an autonomous agent efficiently infer the intent of a demonstrator and provide safety guarantees in the form of high-confidence performance bounds with respect to the demonstrator's intent?

This dissertation addresses this question in four stages:

1. We first provide a theoretical framework and sample efficient Bayesian algorithm for safe imitation learning via high-confidence performance bounds. However, while our proposed approach is efficient in terms of demonstrations, it is computationally inefficient when applied to high-dimensional, complex control problems.

2. To address this computational inefficiency, we next provide reward learning methods that are computationally efficient, scale to high-dimensional control tasks, and can learn policies that perform better than a suboptimal demonstrator. However, these algorithms only learn a point-estimate of the reward function, precluding the kind of robust safety analysis that we desire.

3. To address the problem of safe and efficient imitation learning, we next combine the results from stage 1 and 2 to develop a Bayesian reward learning algorithm that scales to complex visual imitation learning tasks.

4. Finally, we move beyond simply bounding the performance of a policy and consider the problem of robust policy optimization in the imitation learning setting. We present algorithms for robust policy optimization and active policy improvement via risk-aware inverse reinforcement learning.

In the remainder of the introduction we will discuss these stages in more detail and highlight the specific contributions of this dissertation.

We first focus on the problem of safe imitation learning (Brown and Niekum, 2017). In particular, we address the problem of determining high-confidence performance bounds in the inverse reinforcement learning setting, where the true reward function is unknown and must be inferred from samples of expert behavior. We propose a novel Bayesian framework for computing high-confidence probabilistic upper bounds on policy loss when learning from demonstrations (Brown and Niekum, 2018). These bounds are orders of magnitude more sample efficient than prior state-of-the-art bounds, allowing autonomous agents the ability to provide practical high-confidence bounds on generalization performance given small numbers of demonstrations. However, applying our approach to high-dimensional, complex control problems is difficult, due to large computational costs involved when performing standard Bayesian reward function inference (Ramachandran and Amir, 2007). Thus, we next focus on improving the computational efficiency of IRL.

One of the main computational bottlenecks of existing inverse reinforcement learning algorithms is that they require repeatedly fully or partially solving an MDP. This computational burden makes it difficult to scale IRL algorithms to high-dimensional problems. To address this problem, we first propose Trajectory-ranked Reward Extrapolation (T-REX), an inverse reinforcement learning from observation algorithm that can perform sample- and computationally-efficient reward learning via ranked demonstrations (Brown et al., 2019b). T-REX is the first imitation learning algorithm to successfully learn control policies for playing Atari games without access to the ground-truth rewards and learns a reward purely from visual observations without any inference-time access to an MDP.

Furthermore, we demonstrate both theoretically and empirically that rankings (or alternatively, pairwise preferences) over demonstrations not only improve the computational efficiency of IRL, but also allow for better-than-demonstrator performance. To better leverage the advantages of IRL from preferences, we propose two methods for automatically

obtaining pairwise preferences over demonstrations. The first method uses time-stamps as weak supervision for automatically ranking trajectories that are produced by a reinforcement learner. This allows an IRL agent to infer a reward function simply by watching a learner get better at a task. The second method, Disturbance-based Reward Extrapolation (D-REX), first learns a policy from unlabeled demonstrations via supervised learning, and then uses noise injection to automatically generate sets of ranked demonstrations (Brown et al., 2019a). D-REX generalizes T-REX to the standard imitation learning setting where only unlabeled demonstrations are available, while still allowing for better-than-demonstrator performance.

While T-REX and D-REX allow computational- and sample- efficient inverse reinforcement learning, both methods only learn a maximum likelihood estimate of the demonstrator's reward function. However, to effectively reason about safety and uncertainty it is important to have a belief distribution over the demonstrator's true intent. Thus, one of the contributions of this thesis is to combine our work on high-confidence performance bounds for safe imitation learning with our work on efficient IRL from ranked demonstrations to compute high-confidence policy evaluations when learning from high-dimensional visual demonstrations. We first present a general method for computationally-efficient deep Bayesian reward inference that leverages preferences over demonstrations to allow efficient Bayesian inference without requiring an MDP solver in the inner loop (Brown and Niekum, 2019a). Next we utilize self-supervised pre-training of a latent reward function representation to efficiently generate probabilistic worst-case lower bounds on the performance of any policy in the imitation learning setting, where we have no access to ground-truth reward samples (Brown et al., 2020).

Finally, we examine the problem of robust policy optimization and improvement. We first derive a Bayesian robust optimization approach to imitation learning that directly optimizes a policy to balance risk and expected return under a reward function posterior given a fixed set of demonstrations. Next we consider the case where an agent can actively

query for more demonstrations. We use our previous results on high-confidence performance bounds to develop a state-of-the-art active inverse reinforcement learning algorithm that reasons about risk when generating queries, allowing for robust policy improvement. We demonstrate empirically that risk-aware active queries outperform existing active query techniques while also bounding the generalization error of the learner's policy at test time.

1.1 Contributions

In summary, this dissertation makes the following contributions:

1. Formalization of safe imitation learning via high-confidence policy evaluation (Chapter 4).

2. A sample efficient algorithm for obtaining high-confidence performance bounds for imitation learning (Chapter 4).

3. Theoretical results for better-than-demonstrator imitation learning and preference-based inverse reinforcement learning (Chapter 5).

4. A computationally efficient algorithm for reward learning from suboptimal, ranked observations that scales to high-dimensional tasks and can outperform the demonstrator (Chapter 5).

5. Computationally efficient algorithms for learning to extrapolate intention from unlabeled suboptimal demonstrations (Chapter 5).

6. A deep Bayesian inverse reinforcement learning algorithm that scales to complex, high-dimensional tasks (Chapter 6).

7. An algorithm for Bayesian robust imitation learning that optimizes a policy to balance risk and return over a posterior distribution (Chapter 7).

8. An algorithm for risk-aware policy improvement via active inverse reinforcement learning (Chapter 8).

Chapter 2

Background and Related Work

2.1 Reinforcement Learning

Reinforcement learning is the problem of learning how to act in an environment, what actions to take in different situations, in order to maximize a reward signal (Sutton and Barto, 1998). In Section 3.1, we discuss notation in detail. In this section we simply describe some of the basic intuitions behind the reinforcement learning problem before discussing the related problems of imitation learning and inverse reinforcement learning (see Sutton and Barto (1998) for a thorough introduction to reinforcement learning).

In a reinforcement learning problem, the environment is typically modeled as a Markov decision process or MDP (Puterman, 2014; Sutton and Barto, 1998) consisting of states representing the different situations in which the agent may find itself, actions that denote the options available to the agent in each state, the transition dynamics which describe how actions affect state, and a reward function that determines the desirability of taking different actions in different states. For example, for an agent learning to play chess, the states may represent the different possible positions of the pieces on a chess board and the available actions in a state would correspond to the legal moves. Alternatively, for a mobile robot the state could consist of the robot's GPS coordinates and distances to nearby

obstacles as provided by a laser range finder and the actions could be the torques applied to each wheel.

In reinforcement learning, the learning agent's goal is to figure out how to best act, without being told which actions it should take, in order to maximize its long-term, accumulated reward. The reward function provides a scalar feedback about good and bad behavior in the environment. The reward function can be a dense signal, for example, in the mobile robot example above, the robot may be learning to navigate to a goal position and the reward function could be the negative of the distance to the goal. The reward function can also be sparse signal, for example, in the chess example above the agent may receive a reward of +1 for a win, a reward of -1 for a loss, and a reward of 0 otherwise. A reinforcement learning agent seeks to maximize its expected cumulative reward by optimizing a policy. An agent's policy is a mapping from states of the world to actions and determines how the agent will behave in different situations and how well the agent will perform under the reward function.

2.2 Imitation Learning

Imitation learning is the problem of learning a policy from demonstrations of desired behavior. These demonstrations typically consist of sequences of states or sequences of state-action pairs and usually do not include any information about the reward function. Imitation learning can roughly be divided into techniques that use behavioral cloning and techniques that use inverse reinforcement learning.

2.2.1 Behavioral Cloning

Behavioral cloning methods (Pomerleau, 1991; Torabi et al., 2018a) seek to solve the imitation learning problem via supervised learning where the goal is to learn a mapping from states to actions that mimics the demonstrator. While computationally efficient, these methods can suffer from compounding errors. Methods like DAgger (Ross et al., 2011) seek to

avoid this problem by collecting additional state-action pairs from a demonstrator in an online fashion. While methods like DAgger can result in good performance for many problems, they only focus on the "how" of imitation learning and cannot explain "why" the demonstrator entered certain states or took certain actions. We argue in this dissertation that if autonomous agents are to reason about safety, then they must be able to understand and explain the demonstrations by inferring the intent and goals of the demonstrator. Inferring intent is typically done via inverse reinforcement learning which is the topic of the next section.

2.2.2 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) (Ng and Russell, 2000) seeks to solve the imitation learning problem by first estimating a reward function that makes the demonstrations appear optimal, and then running reinforcement learning (Sutton and Barto, 1998) on the inferred reward function to learn a policy. Classical approaches to IRL repeatedly alternate between reward estimation and full policy optimization (Abbeel and Ng, 2004; Ramachandran and Amir, 2007; Ziebart et al., 2008). Bayesian IRL (Ramachandran and Amir, 2007) generates samples from the posterior distribution of likely reward functions given the demonstrations, whereas other methods seek a single estimate of the reward function that matches the demonstrator's state occupancy (Abbeel and Ng, 2004), often while also seeking to maximize the entropy of the resulting policy (Ziebart et al., 2008).

Modern, deep learning approaches to inverse reinforcement learning are typically based on a maximum entropy framework (Finn et al., 2016) or an occupancy matching framework (Ho and Ermon, 2016) and are related to Generative Adversarial Networks (Goodfellow et al., 2014). These methods scale to complex control problems by iterating between a few steps of reward learning and a few steps of policy learning. In Section 5.2 we propose an IRL algorithm which uses preferences labels over an initial set of demonstrations to efficiently learn a reward function from visual observations via supervised learning

without requiring fully or partially solving an MDP (Brown et al., 2019b). While existing deep IRL methods have shown recent success, existing methods typically only return a point estimate of the reward function, precluding the rich uncertainty and robustness analysis possible with a full Bayesian approach. In Chapter 6, we use self-supervised deep learning to scale Bayesian IRL to complex control problems with visual observations, allowing an agent to sample from the full posterior distribution over reward functions given the demonstrations.

2.2.3 Learning from Observation

Recently there has been a shift towards imitation learning from observations, where the actions taken by the demonstrator are unobserved. Torabi et al. (2018a) propose a state-of-the-art model-based approach to perform behavioral cloning from observation. Sermanet et al. (2018) and Liu et al. (2018) propose methods to learn directly from a large corpus of videos containing multiple view points of the same task. Yu et al. (2018) and Goo and Niekum (2019) propose meta-learning-from-observation approaches that can learn from a single demonstration, but require training on a wide variety of similar tasks. Henderson et al. (2018) and Torabi et al. (2018b) extend Generative Adversarial Imitation Learning (Ho and Ermon, 2016) to remove the need for action labels. However, inverse reinforcement learning methods based on Generative Adversarial Networks (Goodfellow et al., 2014) are difficult to train and are difficult to scale to high-dimensional imitation learning tasks such as Atari (Tucker et al., 2018). In Section 5.2 we present one of the first imitation learning from observation methods that scales to high-dimensional visual tasks such as Atari and can outperform the demonstrator.

2.2.4 Better-than-Demonstrator Imitation Learning

While imitation learning has grown increasingly popular in recent years (Argall et al., 2009; Gao et al., 2012; Osa et al., 2018; Arora and Doshi, 2018), little work has addressed

the problem of achieving better-than-demonstrator performance. One notable exception is when using demonstrations to assist reinforcement learning. When ground-truth rewards are known, it is common to initialize a policy using demonstrations and then improve this policy using reinforcement learning (Kober and Peters, 2009; Taylor et al., 2011; Hester et al., 2018; Gao et al., 2018; Sarafian et al., 2018). However, designing good reward functions for reinforcement learning can be difficult and can easily lead to unintended behaviors (Ng et al., 1999; Amodei et al., 2016).

Rather than relying on a hand-crafted reward function, this dissertation focuses on using imitation learning to estimate a demonstrator's reward function. While there has been some work on imitation learning from suboptimal demonstrations, prior approaches often either require poor demonstrations to be manually clustered into those that overshoot and undershoot a goal (Grollman and Billard, 2011) or require clusters of optimal and suboptimal demonstrations (Shiarlis et al., 2016). Other methods are robust to unlabeled, poor demonstrations, but require the majority of the demonstrations to come from an expert in order to correctly identify which demonstrations are anomalous (Zheng et al., 2014; Choi et al., 2019). Syed and Schapire (2007) prove that knowledge about which features contribute positively or negatively to the true reward allows an apprenticeship policy to outperform the demonstrator. However, their approach requires hand-crafted, linear features, knowledge of the true signs of the rewards features, and repeatedly solving an MDP in the inner loop.

In Section 5.2 of Chapter 5, we propose an algorithm for efficiently learning a better-than-demonstrator policy via pre-ranked observations. In Section 5.3 we extend this work by proposing two methods for learning to exceed the performance of a demonstrator. We first examine the case where we learn from a learner (Jacq et al., 2019). In this case, we assume the demonstrator is improving at a task so the temporal ordering of trajectories provides weak preference labels. Next we propose a reward learning algorithm that works with any set of unlabeled demonstrations and uses noise injection to automatically generate

ranked demonstrations. Prior work on imitation learning has investigated the use of random or noisy trajectories. Boularias et al. (Boularias et al., 2011) and Kalakrishnan et al. (Kalakrishnan et al., 2013) use uniformly random and locally perturbed trajectories, respectively, to estimate the partition function for Maximum Entropy IRL (Ziebart et al., 2008). Both methods seek a linear combination of predefined features such that the returns of the demonstrations are maximized with respect to the random trajectories. These methods can be seen as a special case of our proposed method, where only one level of noise is used and where the reward function is represented as a linear combination of known features.

Disturbances for Augmenting Robot Trajectories (DART) (Laskey et al., 2017) is a recently proposed behavioral cloning approach that adds noise during demonstrations to collect a richer set of state-action pairs for behavioral cloning. DART avoids the problem of compounding error that is common to most behavioral cloning approaches by repeatedly requesting and perturbing new demonstrations. Instead of repeatedly collecting perturbed trajectories from the demonstrator, our approach in Section 5.3.2 collects a small number of initial demonstrations, run behavioral cloning once, and then inject varying amounts of noise into the cloned policy. This automatically creates a large set of ranked demonstrations for reward learning without requiring a human to provide ranking labels. Amin et al. (Amin and Singh, 2016) proved that a logarithmic number of demonstrations from a family of MDPs with different transition dynamics is sufficient to resolve reward ambiguity in IRL. Automatically-generating ranked trajectories via noise injection, can be seen as an efficient heuristic for generating demonstrations under different transition dynamics.

2.3 Safety

There is a growing interest in making AI and machine learning systems safe and well-behaved (Garcıa and Fernández, 2015; Amodei et al., 2016; Thomas et al., 2019). In the next sections we briefly cover research on safety that is most relevant to this dissertation proposal.

2.3.1 Reinforcement Learning

Because reinforcement learning agents often learn via trial and error, safety is an important problem, as there is often the possibility of taking actions that may be risky and could potentially lead to catastrophic consequences. Safety has been extensively studied within the reinforcement learning community (see (Garcıa and Fernández, 2015) and (Amodei et al., 2016) for good surveys). Most safe reinforcement learning approaches typically either focus on safe exploration or on optimizing an objective other than expected return. Recently, objectives based on financial measures of risk such as value at risk (Jorion, 1997) and conditional value at risk (Rockafellar and Uryasev, 2000) have been shown to provide tractable and useful risk-sensitive measures of performance (Tamar et al., 2015; Chow et al., 2015). Other related work on safe reinforcement learning has focused finding robust policies using Bayesian ambiguity sets (Petrik and Russell, 2019) and on obtaining high-confidence bounds on the performance of an evaluation policy by using data generated by a different behavior policy (Thomas et al., 2015b; Hanna et al., 2017). The research presented in chapters 4 and 6 of this dissertation complements existing work on high-confidence off policy evaluation by proposing solutions to high-confidence off-policy evaluation in the imitation learning setting, where samples of rewards are not observed and the demonstrator's policy (the behavioral policy) is unknown. In Chapter 7 we present a robust imitation learning algorithm based on Conditional Value at Risk that complements similar work in the reinforcement learning setting (Tamar et al., 2015; Chow et al., 2015; Tang et al., 2020). Finally, in Chapter 8 we consider the problem of using high-confidence policy evaluation for high-confidence policy improvement in the imitation learning setting. This work complements existing work on high-confidence policy improvement in the reinforcement learning setting (Thomas et al., 2015a).

2.3.2 Safe Imitation Learning

Recently, there has been growing interest in safe imitation learning. Zhang and Cho (2017) propose SafeDAgger a variant of DAgger that predicts in which states the novice policy will have a large action difference from the expert policy. Control is given the the expert policy only if the predicted action difference of the novice is above some hand-tuned parameter, τ. Other work has focused on risk-sensitive generative adversarial imitation learning. Lacotte et al. (2019) propose an imitation learning algorithm that seeks to find a specific policy that matches the tail risk of the expert and is indistinguishable from the demonstrations. Hadfield-Menell et al. (2017) propose to learn a distribution over reward functions given a single sample of a reward function. They then optimize a policy that is robust to this distribution but do not provide performance bounds. In chapters 4 and 6 we discuss how to provide explicit high-confidence safety bounds that work for any imitation learning policy.

An important challenge in inverse reinforcement learning (IRL) is dealing with ambiguity over the reward function (Ng and Russell, 2000; Ziebart et al., 2008), since there are usually an infinite number of reward functions that are consistent with a set of demonstrations (Ng and Russell, 2000). Problems with such ambiguous parameters can be solved using robust optimization techniques, which compute the best policy for the worst rewards consistent with the demonstrations (Ben-Tal et al., 2009). Indeed, many IRL methods optimize policies for the worst-case rewards (Huang et al., 2018; Hadfield-Menell et al., 2017; Ho and Ermon, 2016; Syed et al., 2008). This optimization for the worst-case parameter values is well known to lead to overly conservative solutions across many domains (Delage and Mannor, 2010; Russell and Petrik, 2019; Iancu and Trichakis, 2014). In Chapter 7, we rely on coherent measures of risk to represent the trade-off between the average and worst-case performance (Artzner et al., 1999; Shapiro et al., 2014; Follmer and Schied, 2011). Similar approaches to parameter uncertainty, also known as epistemic uncertainty, have been referred to as soft-robustness in earlier work (Derman et al., 2018; Ben-Tal et al., 2010) but have not been studied in the context of IRL.

RS-GAIL and related algorithms (Majumdar et al., 2017; Lacotte et al., 2019; Santara et al., 2017) mitigate risk in IRL but assume risk-averse experts and focus on optimizing policies that match the risk-aversion of the demonstrator; however, unlike our results in chapters 4 and 6, these methods do not provide high-confidence bounds on performance. Furthermore, these methods focus on the uncertainty induced by transition probabilities, also known as aleatoric risk. The challenges in this area are very different and there is no obvious way to adapt risk-averse IRL to our Bayesian robust setting described in Chapter 7 where we seek to be robust to epistemic risk rather than seeking to match the risk of the demonstrator. RBIRL (Zheng et al., 2014) seeks to infer a posterior distribution that is robust to small numbers of bad demonstrations, but does not address robust policy optimization with respect to ambiguity in the learned posterior. While the safe imitation learning methods we describe in chapters 4, 7, and 8 are not explicitly robust to bad demonstrations, the methods we propose in this dissertation can make use of any posterior distribution over reward functions. Thus, they can be easily be extended to use posteriors generated from methods like RBIRL (Zheng et al., 2014) in order to be robust to small numbers of poor demonstrations. The methods described in chapters 5 and 6 of this dissertation are robust to bad demonstrations by means of learning a reward function via accurate preference rankings. Finally, FPL-IRL (Huang et al., 2018), LPAL (Syed et al., 2008), GAIL (Ho and Ermon, 2016) only optimize the policy for a (regularized) worst-case realization of the rewards. These approaches explicitly try to match the state or state-action occupancies of the demonstrator, precluding significantly better-than-demonstrator performance.

2.3.3 High-Confidence Performance Bounds for Imitation Learning

Early work on inverse reinforcement learning also provides high-confidence performance bounds. Abbeel and Ng (2004) and Syed and Schapire (2007) give probabilistic Hoeffding-style bounds on how many demonstrations their algorithms require to guarantee a policy with expected return within epsilon of the expected return of the demonstrator's policy.

However, as shown in Chapter 4, these theoretical bounds are too loose to be useful in practice. Methods such as DAgger (Ross et al., 2011) provide regret bounds on a policy obtained via behavioral cloning, but require constant human supervision during policy learning and are not able to recover a reward function, precluding performance bounds for arbitrary evaluation policies. In Chapter 4, we propose the first sample-efficient, high-confidence bound on the policy loss of any evaluation policy with respect to the optimal policy under the demonstrator's true reward function. In Chapter 6 we extend high-confidence policy evaluation to high-dimensional visual imitation learning tasks.

2.4 Learning from Human Feedback

Much work has been done on interactive machine learning, where a learning system has access to feedback from a human. In this section we briefly describe some of the most relevant work to this dissertation.

2.4.1 Interactive Reinforcement Learning

Leveraging interactive human feedback is an efficient and popular way to assist and guide a learning agent. The TAMER framework proposed by (Knox and Stone, 2009) and the COACH framework proposed by (MacGlashan et al., 2017) both cast interactive learning as a reinforcement learning (RL) problem. TAMER interprets human feedback as uniform reward signals while COACH argues that human feedback is policy dependent and should be treated as advantage signals. However, in both of these algorithms the learning agent passively receives critiques without actively acting or querying for help in a way that facilitates better understanding of the human's intent. In Chapter 8 we propose a safe inverse reinforcement learning method that pinpoints situations where the learner may perform poorly and then actively asks for feedback from the demonstrator about what to do in these high-risk situations. We discuss different forms of active learning in the next section.

2.4.2 Active Inverse Reinforcement Learning

In contrast to human-guided RL, there has also been significant work on active IRL approaches where an agent explicitly queries the demonstrator for help. Most active IRL algorithms use a Bayesian approach to systematically addresses the ambiguity in reward learning (Cohn et al., 2011; Cui and Niekum, 2018; Lopes et al., 2009; Sadigh et al., 2017). Cohn et al. (2011) propose a risk-neutral approach that generates active action in order to increase the agent's expected discounted reward. Similar to Cohn et al. (2011), Cui and Niekum (2018) directly compute expected information gain over the distribution of reward functions per state-action pair to maximally reduce uncertainty over the reward distribution. However, both of these methods requires significantly more computation than the approach we present in Chapter 8 and do not provide an upper bound on policy loss. To generate a single active query, Cohn et al. (2011) and Cui and Niekum (2018) repeatedly estimate the expected value of information for every state-action pair in the MDP. This renders their approach computationally intractable for continuous spaces of reward functions, since generating a single active query using their approach requires running Bayesian IRL (an already computationally demanding algorithm) for each state-action pair in the entire MDP. In contrast, our risk-aware active IRL approach in Chapter 8 only requires running Bayesian IRL once per active query. The work of (Lopes et al., 2009) reasons about state-wise policy entropy over the posterior distribution of policies given demonstrations and asks for demonstrations at states with the highest action entropy. The state-wise policy entropy is ignorant of the actual policy that will be used for evaluation and does not take the values of states into account.

Active IRL approaches typically do not explicitly use the performance of the robot's learned policy to generate queries. One of the primary reasons for this is that, prior to this dissertation, practical methods for estimating the performance of a policy learned from demonstrations when the ground-truth reward function is unknown, have not existed. In Chapter 4, we present a practical method for computing accurate, tight bounds on the per-

formance loss of any imitation policy. In Chapter 8, build on the results of Chapter 4 to develop a novel approach to active learning that focuses directly on minimizing the generalization error of the robot's learned policy. We demonstrate that this approach achieves state-of-the-art performance when compared with other active IRL approaches while also enabling a learning agent to know when it has received enough demonstrations to guarantee good performance with high-confidence.

2.4.3 Preference-Based Learning

Chapters 5 and 6 make use of preference rankings over demonstrations and can be seen as forms of preference-based policy learning (Akrour et al., 2011) and preference-based IRL (PBIRL) (Wirth et al., 2016; Sugiyama et al., 2012), which both seek to optimize a policy based on preference rankings over demonstrations. However, most existing approaches to preference-based policy learning only consider reward functions that are linear in hand-crafted features and have not studied extrapolation capabilities. In contrast, the methods we propose in chapters 5 and 6 use deep learning to automatically learn from raw visual features and exhibit the ability to significantly extrapolate beyond the performance of the demonstrator. For a more complete overview survey of preference-based reinforcement learning, see the survey by (Wirth et al., 2017). Other preference-based learning methods (Burchfiel et al., 2016; El Asri et al., 2016) have proposed the use of quantitatively scored trajectories as opposed to qualitative pairwise preferences over demonstrations. Our methods can also be applied to the case of quantitative scores by simply extracting appropriate pairwise preferences labels.

In active preference learning (Eric et al., 2008), the learning agent synthesizes preference queries for the demonstrator to label. Sadigh et al. (2017) and Christiano et al. (2017) propose reward learning approaches that use active learning to collect pairwise preferences labels. Ibarz et al. (2018) and Palan et al. (2019) combine demonstrations with active preference learning during policy optimization. Rather than collecting pairwise pref-

erences via active queries, in Chapter 5 we propose Trajectory-ranked Reward Extrapolation (T-REX), an algorithm that uses a set of pre-ranked demonstrations to learn a reward function. Furthermore, in Chapter 5, we demonstrate that ranking-based imitation learning approach are applicable even in cases where human rankings are unavailable. Finally, in Chapter 6, we provide the first preference-based Bayesian reward inference method that scales to high-dimensional complex imitation learning tasks such as learning to play Atari games (Bellemare et al., 2013) from observation without access to the game score.

Chapter 3

Notation and Preliminaries

In this chapter we introduce the notation that we will use throughout most of this dissertation. For a more in depth treatment of Markov decision processes and reinforcement learning, we recommend Puterman (2014) and Sutton and Barto (1998).

3.1 Markov Decision Processes

A Markov decision process (MDP) is defined as a tuple $\langle S, A, T, R, \gamma, S_0 \rangle$ where S is the set of states, A is the set of actions, $T : S \times A \times S \to [0, 1]$ is the transition function, $R : S \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1)$ is the discount factor, and S_0 is the initial state distribution.

A policy π is a mapping from states to a probability distribution over actions. The value of a policy π under reward function R is the expected return of that policy and is denoted as $V_R^\pi = \mathbb{E}_{s_0 \sim S_0}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi]$. The value of executing policy π starting at state $s \in S$ is defined as $V_R^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s]$. Given a reward function R, the Q-value of π of a state-action pair (s, a) is defined as the expected return achieved

by starting in state s, taking action a, and then following π thereafter:

$$Q_R^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s, a_0 = a]. \tag{3.1}$$

We denote $V_R^* = \max_\pi V_R^\pi$ and $Q_R^*(s,a) = \max_\pi Q_R^\pi(s,a)$.

3.2 Linear Reward Functions

As is common in the literature (Abbeel and Ng, 2004; Ziebart et al., 2008; Sadigh et al., 2017; Pirotta and Restelli, 2016), we will often assume that the reward function can be expressed as a linear combination of features, so that $R(s) = w^T\phi(s)$ where $w \in \mathbb{R}^k$ is the k-dimensional vector of feature weights. Thus, we can write the value of a policy as

$$V_R^\pi \;=\; \mathbb{E}_{s_0 \sim S_0}[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s_t) \mid \pi] \;=\; w^T \Phi(\pi), \tag{3.2}$$

where $\Phi(\pi) = \mathbb{E}_{s_0 \sim S_0}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi]$ are the expected feature counts. Note that this does not affect the expressiveness of the reward function since ϕ can be a non-linear function. Given ϕ, the reward function is fully specified by the feature weights w. Thus, in the case of linear reward functions, we will refer to the feature weights w and the reward function R interchangeably.

3.3 Bayesian Inverse Reinforcement Learning

In inverse reinforcement learning (IRL) Ng and Russell (2000), we are given an MDP without a reward function, denoted as MDP\R. Given a set of demonstrations,

$$D = \{(s_1, a_1), \ldots, (s_m, a_m)\}, \tag{3.3}$$

, consisting of state-action pairs, Bayesian IRL (Ramachandran and Amir, 2007) seeks to estimate the posterior over reward functions given demonstrations, $P(R|D) \propto P(D|R)P(R)$. Bayesian IRL assumes a Boltzman-rational demonstrator that executes the following policy

$$\pi_R^\beta(a|s) = \frac{e^{\beta Q_R^*(s,a)}}{\sum_{b \in \mathcal{A}} e^{\beta Q_R^*(s,b)}}, \tag{3.4}$$

in which R is the true reward function of the demonstrator, and $\beta \in [0, \infty)$ represents the confidence that the demonstrator is acting optimally. Under the assumption of Boltzman rationality, the likelihood of a set of demonstrated state-action pairs, $D = \{(s,a) : (s,a) \sim \pi_D\}$, given a specific reward function hypothesis R, can be written as

$$P(D|R) = \prod_{(s,a) \in D} \pi_R^\beta(a|s) = \prod_{(s,a) \in D} \frac{e^{\beta Q_R^*(s,a)}}{\sum_{b \in \mathcal{A}} e^{\beta Q_R^*(s,b)}}. \tag{3.5}$$

where $Q_R^*(s,a)$ is the optimal Q-value function for reward R, and β is a parameter representing the confidence in the demonstrator's optimality. Equation 3.5 gives greater likelihood to reward functions for which the actions taken by the expert result in higher expected rewards than the alternative actions.

The softmax distribution over actions is commonly used as a likelihood function in IRL (Babes et al., 2011; Levine et al., 2011; Michini and How, 2012a; Rothkopf and Ballard, 2013) and has been empirically shown to be an effective model of human behavior, enabling accurate learning from human demonstrations (Lopes et al., 2007; Kim and Pineau, 2016) and prediction of human actions (Baker et al., 2009; Karasev et al., 2016). Shah et al. (2019) proposed a method for learning a model of demonstrator biases; however, learning these models requires large amounts of meta-learning on tasks with known rewards, and can still fail to outperform the assumption of Boltzman rationality, even if the actual demonstrations follow a vastly different planning model. In Chapters 4, 7, and 8 we model the demonstrator via Boltzman rationality, although we note that if a high-quality model of demonstrator behavior is available, then it can easily be inserted into our

proposed approach. In Chapters 5 and 6 we explicitly examine the case where the demonstrator's behavior may be highly and systematically suboptimal, but where the demonstrator can provide preferences over trajectories.

The choice of the prior distribution $P(R)$ allows domain knowledge to be inserted into the IRL algorithm. Ramachandran and Amir (2007) give several possibilities such as a uniform, Gaussian, or Beta prior. Choi and Kim (2011) showed that many standard IRL algorithms can be transformed into an equivalent Bayesian IRL algorithm by selecting the appropriate likelihood and prior. Thus, our proposed performance bound can be easily extended to use alternative likelihoods and priors that match different assumptions and preferences found in the IRL literature. Unless otherwise specified we will typically assume a uniform prior throughout this dissertation.

Bayesian IRL uses Markov chain Monte Carlo (MCMC) sampling to sample from the posterior $P(R|D)$ (Ramachandran and Amir, 2007; Brown and Niekum, 2018). Feature weights are sampled according to a proposal distribution, and for each sample the MDP is solved to obtain the sample's likelihood and determine the transition probabilities within the Markov chain. An estimate of the expert's reward function can be found by averaging the feature weights in the chain to obtain the mean reward function (Ramachandran and Amir, 2007) or by using the maximum a posteriori (MAP) estimate (Choi and Kim, 2011). Some of the advantages of Bayesian IRL, compared to many other IRL algorithms, are (1) it finds a distribution over likely reward functions, (2) the set of demonstrations can contain partial demonstrations or even non-contiguous state action pairs, and (3) it allows domain knowledge in the form of a prior.

Chapter 4

High-Confidence Performance Bounds for Safe Imitation Learning

Imitation learning has potential applications in many settings such as manufacturing, home and hospital care, and autonomous driving. In these types of real-world settings it is important, and perhaps critical, to provide performance bounds on an agent's learned policy. For example, consider a hospital assistant robot that has learned from demonstrations how to lift a patient out of bed. Before deploying this learned policy, we would want to provide a high-confidence bound on the difference in performance between the robot's learned policy and the optimal policy under the expert's reward. If this bound on policy loss is too high, then the robot could request additional demonstrations until, with high confidence, its policy loss with respect to the optimal policy is within some allowable error margin.

In this chapter, we focus on the problem of high-confidence policy evaluation. This chapter comprises contributions 1 and 2 of this dissertation. Contribution 1 is a formalization of safe imitation learning via high-confidence policy evaluations without access to the ground-truth reward function. Contribution 2 is a sample efficient method for calculating high-confidence performance bounds on the α-quantile worst-case policy loss in the imitation learning setting—where the true reward function is unknown and only samples

of expert behavior are given. To the best of our knowledge, this chapter presents the first method to provide sample-efficient, risk-aware confidence bounds on the performance of a policy under an unknown reward function, as is the case when learning from demonstrations.[1]

While the methods described in this chapter are highly sample-efficient, in terms of the number of demonstrations, they require an MDP solver in the inner-loop which makes it difficult to scale these approaches to complex, high-dimensional problems. In Chapter 5 we will address this inefficiency by proposing reward inference algorithms that do not require an MDP solver in the inner-loop, allowing them to scale to complex control tasks. Finally, in Chapter 6 we will combine the work presented in this chapter and the efficient reward inference algorithms proposed in Chapter 5, to develop a tractable solution to the high-confidence policy evaluation problem for high-dimensional, complex imitation learning tasks.

In this chapter, we propose a sampling method based on Bayesian inverse reinforcement learning that uses demonstrations to determine practical high-confidence upper bounds on the α-worst-case difference in expected return between any evaluation policy and the optimal policy under the expert's unknown reward function. We evaluate our proposed bound on both a standard grid navigation task and a simulated driving task and achieve tighter and more accurate bounds than a feature count-based baseline. We also give examples of how our proposed bound can be utilized to perform risk-aware policy selection and risk-aware policy improvement. Because our proposed bound requires several orders of magnitude fewer demonstrations than existing high-confidence bounds, it is the first practical method that allows agents that learn from small numbers of demonstration to express confidence in the quality of their learned policy.

[1] This chapter contains work that was done in collaboration with Scott Niekum and was previously published at the AAAI Fall Symposium 2017 (Brown and Niekum, 2017) and at AAAI 2018 (Brown and Niekum, 2018).

4.1 High-Confidence Policy Evaluation for Imitation Learning

Before detailing our approach, we first formalize the problem of high-confidence policy evaluation for imitation learning. We assume access to a Markov decision process without a reward function (MDP\R), an evaluation policy π_{eval}, a set of demonstrations, $D = \{\tau_1, \ldots, \tau_m\}$, in which τ_i can either be a complete or partial trajectory comprised of states or state-action pairs, a confidence level δ, and risk metric $g : \Pi \times \mathcal{R} \rightarrow \mathbb{R}$, in which $\mathcal{R}$ denotes the space of reward functions and Π is the space of all policies.

The ***High-Confidence Policy Evaluation problem for Imitation Learning*** (HCPE-IL) is to find a high-confidence upper bound $\hat{g} : \Pi \times \mathcal{D} \rightarrow \mathbb{R}$ such that

$$\Pr(g(\pi_{\text{eval}}, R^*) \geq \hat{g}(\pi_{\text{eval}}, D)) \leq 1 - \delta, \tag{4.1}$$

in which R^* denotes the demonstrator's true reward function and $\mathcal{D}$ denotes the space of all possible demonstration sets. HCPE-IL takes as input an evaluation policy π_{eval}, a set of demonstrations D, and a risk metric, g, which evaluates a policy under a reward function. The goal of HCPE-IL is to return a high-confidence upper bound $\hat{g}$ on the performance statistic $g(\pi_{\text{eval}}, R^*)$.

Note that this problem setting is significantly different from high-confidence off-policy evaluation problem in reinforcement learning (Thomas et al., 2015b), which we denote as HCOPE-RL. In HCOPE-RL the behavior policy is usually known and the demonstrations from the behavior policy contain ground-truth reward samples. In HCPE-IL, the demonstrator's policy π_{R^*} is unknown and the demonstration data from π_{R^*} consists only of either states or state-action pairs—no samples from a ground-truth reward signal are available. Given this general formalism for safe imitation learning, we now describe a specific instantiation of the HCPE-IL problem and our proposed solution.

4.2 High-Confidence Bounds on Policy Loss

Given the definition in the previous section, we now present a sample efficient algorithm for solving this problem. In particular, we will focus on using policy loss as the risk metric g. We define policy loss using the *Expected Value Difference* (EVD) of π_{eval} under the true reward R^*, defined as

$$\text{EVD}(\pi_{\text{eval}}, R^*) = V_{R^*}^* - V_{R^*}^{\pi_{\text{eval}}}. \tag{4.2}$$

We use EVD because it is a natural way to measure the performance difference between two policies and it is a common metric for evaluating IRL algorithms (Ramachandran and Amir, 2007; Levine et al., 2011; Choi and Kim, 2011; Wulfmeier et al., 2015). Note that the evaluation policy can be any policy, including a hand-tuned policy or a policy learned through reinforcement learning on a different task with a known reward function; however, it is often the case that the most natural form of the evaluation policy is a policy learned from the demonstrations, D.

As is common in the literature (Abbeel and Ng, 2004; Ziebart et al., 2008; Sadigh et al., 2017), we assume that the reward function can be expressed as a linear combination of features, so that $R(s) = w^T\phi(s)$ where $w \in \mathbb{R}^k$ is the k-dimensional vector of feature weights. Thus, we can write the value of a policy as

$$V_R^\pi \;=\; \mathbb{E}_{s_0\sim S_0}[\sum_{t=0}^{\infty}\gamma^t w^T\phi(s_t) \mid \pi] \;=\; w^T\Phi_\pi, \tag{4.3}$$

where $\Phi_\pi \;=\; \mathbb{E}_{s_0\sim S_0}[\sum_{t=0}^{\infty}\gamma^t\phi(s_t) \;\mid\; \pi]$ are the expected feature counts. Note that this does not affect the expressiveness of the reward function since $\phi(s)$ can be a highly non-linear function of the state s. In this chapter we will assume that ϕ is given. While ϕ may be a hand-crafted set of features, ϕ can also be learned from raw observations. In Chapter 6 we present a deep learning method for automatically discovering the features of a linear reward function via a self-supervised pre-training step that can be performed before imitation learning. Given ϕ, the reward function is fully specified by the feature weights w.

Thus, we refer to the feature weights w and the reward function R interchangeably. Given a linear reward function, we can express the EVD as the following dot product:

$$\text{EVD}(\pi_{\text{eval}}, R^*) = V^*_{R^*} - V^{\pi_{\text{eval}}}_{R^*} = w^T(\Phi_{\pi*} - \Phi_{\pi_{\text{eval}}}) \tag{4.4}$$

The main goal of this chapter is to bound the difference in expected return between any evaluation policy π_{eval} and π^*, the policy that is optimal with respect to the demonstrator's reward R^*. However, because an optimal policy is invariant to any non-negative scaling of the reward function, bounding EVD is ill-posed, as we can multiply the feature weights w by any $\beta > 0$ to scale EVD to be anywhere in the range $[0, \infty)$. To avoid this scaling issue we make the common assumption that $\|w\|_1 = 1$ (Syed and Schapire, 2007; Pirotta and Restelli, 2016). Note, that this assumption only eliminates the trivial all-zero reward function as a potential solution—all other reward functions can be appropriately normalized. While setting $\|w\|_1 = 1$ eliminates the invariance to scaling factors and bounds the magnitude of the EVD, there can still be infinitely many rewards that induce any optimal policy, resulting in infinitely many possible values of $\text{EVD}(\pi_{\text{eval}}, R^*)$. Thus, to obtain an upper bound on $\text{EVD}(\pi_{\text{eval}}, R^*)$ we need to address this uncertainty. As we show in the next section, one way to address this uncertainty over the demonstrator's true reward is to compute an absolute worst-case policy loss bound using feature counts.

4.3 Worst-Case Bound

In this section, we derive a simple worst-case bound based on feature counts that we use as a baseline. As a reminder, we use the notation $\Phi_\pi = \mathbb{E}_{s_0 \sim S_0}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi]$ to represent the expected feature counts of policy π.

If we can assume that $R = w^T\phi(s)$, $\phi(s) : S \to [0, 1]^k$, $\|w\|_1 \leq 1$, and that we know the demonstrator's expected feature counts $\Phi_{\pi_{\text{demo}}}$, then given any evaluation policy

π_{eval}, the following is true:

$$V_R^{\pi_{\text{demo}}} - V_R^{\pi_{\text{eval}}} = \left| \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi_{\text{demo}} \big] - \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi_{\text{eval}} \big] \right| \tag{4.5}$$

$$= \left| \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s_t) \mid \pi_{\text{demo}} \big] - \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s_t) \mid \pi_{\text{eval}} \big] \right| \tag{4.6}$$

$$= \left| w^T \left(\mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi_{\text{demo}} \big] - \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi_{\text{eval}} \big] \right) \right| \tag{4.7}$$

$$= w^T (\Phi_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}}) \tag{4.8}$$

$$\leq \|w\|_1 \|\Phi_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}}\|_\infty \tag{4.9}$$

$$\leq \|\Phi_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}}\|_\infty. \tag{4.10}$$

which uses the fact that $R(s) = w^T \phi(s)$, Hölder's inequality, and the assumption that $\|w\|_1 \leq 1$. This proof follows that of Abbeel and Ng (2004), except that our bound is strictly tighter as a result of using the infinity norm rather than the 2-norm.

If π_{demo} is an optimal policy with respect to the demonstrator's reward function, R^*, then we have that

$$\begin{aligned} \text{EVD}(\pi_{\text{eval}}, R^*) &= V_{R^*}^{\pi^*} - V_{R^*}^{\pi_{\text{eval}}} && (4.11) \\ &= V_{R^*}^{\pi_{\text{demo}}} - V_{R^*}^{\pi_{\text{eval}}} && (4.12) \\ &= |w^T (\Phi_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}})| && (4.13) \\ &\leq \|\Phi_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}}\|_\infty && (4.14) \end{aligned}$$

and thus $\|\Phi_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}}\|_\infty$ gives an upper bound on $\text{EVD}(\pi_{\text{eval}}, R^*)$. In practice, we typically do not know μ^*, but we can use demonstrated trajectories to estimate of the demonstrator's expected feature counts as

$$\hat{\mu}^* = \frac{1}{|D|} \sum_{i=1}^{|D|} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}), \tag{4.15}$$

where i indexes over the trajectories and t over the state sequence contained in each demonstrated trajectory. We define the empirical *worst-case feature count bound* as

$$\text{WFCB}(\pi_{\text{eval}}, D) = \|\hat{\Phi}_{\pi_{\text{demo}}} - \Phi_{\pi_{\text{eval}}}\|_\infty. \tag{4.16}$$

Note that this bound is only guaranteed to be an upper bound on $\text{EVD}(\pi_{\text{eval}}, R^*)$ if π_{demo} is optimal and if we have sufficient demonstrations such that $\hat{\Phi}_{\pi_{\text{demo}}} = \Phi_{\pi_{\text{demo}}}$. In practice, we may only have a small number of demonstrations, in which case we can compute the probability that WFCB is an upper bound on $\text{EVD}(\pi_{\text{eval}}, R^*)$; however, as we will discuss later, obtaining a high-confidence upper bound via the empirical estimate of the expert's feature counts, $\hat{\Phi}_{\pi_{\text{demo}}}$, usually requires an extremely large number of demonstrations (Abbeel and Ng, 2004; Syed and Schapire, 2007). Note also that the WFCB bound does not work with partial demonstrations and that it is based on an adversarial reward function that may be extremely unlikely given the demonstrations.

4.4 EVD Value-at-Risk Bound

The worst-case feature count bound described in the previous section only requires sampled trajectories from the expert, but ignores much of the structure of the problem and the specific actions taken by the demonstrator—giving a worst-case bound that will likely be overly pessimistic and result in loose bounds. Thus, rather than focusing on absolute worst-case, we instead propose to focus on computing a probabilistic upper bound on the α-worst-case value of $\text{EVD}(\pi_{\text{eval}}, R)$, where $R \sim P(R|D)$. Our goal in this section is to obtain a high-confidence probabilistic worst-case bound that focuses on likely reward functions given the demonstrations.

The α-worst-case value of a random variable is often referred to in finance as the α-Value at Risk (Jorion, 1997). We use the notation of Tamar et al. (2015) and formally

define the α-*Value-at-Risk* (α-VaR) of a random variable Z as

$$\nu_\alpha(Z) = F_Z^{-1}(\alpha) = \inf\{z : F_Z(z) \geq \alpha\} \tag{4.17}$$

where $\alpha \in (0, 1)$ is the quantile level and $F_Z(z) = Pr(Z \leq z)$ is the cumulative distribution function of Z. The parameter α defines the sensitivity to risk, while $(1 - \delta)$ represents our confidence in our estimate of the α-VaR. Thus, while $(1 - \delta)$ is typically always high (e.g., 0.95), α can take on a range of values depending on the possibility of catastrophic failure in the domain and the risk-aversion of the end-user. In practice, $\alpha \geq 0.9$ is commonly used for VaR applications (Jorion, 1997). We can now state the specific instantiation of the HCPE-IL problem that we address in this chapter:

Given an MDP\R, any evaluation policy π_{eval}, and a set of demonstrations, D, find a $(1 - \delta)$ confidence upper bound on $\nu_\alpha(\text{EVD}(\pi_{\text{eval}}, R))$, where $R \sim P(R\|D)$.

We seek to directly compute a high-confidence bound on the α-Value at Risk of the $\text{EVD}(\pi_{\text{eval}}, R^*)$ for any given evaluation policy π_{eval}. Instead of bounding EVD using an empirical estimate of the demonstrator's feature counts, we seek to compute a high-confidence bound using samples from the posterior distribution $\text{EVD}(\pi_{\text{eval}}, R)$, where $R\ P(R|D)$. Eliminating the need to accurately estimate the demonstrator's feature counts $\hat{\Phi}_{\pi_{\text{demo}}}$ is desirable for two main reasons. The first reason is that removing the need for expected feature counts enables us to formulate a high-confidence bound that works well with partial, noisy demonstrations. This is because EVD compares the evaluation policy against the optimal policy for reward R, not the actual states visited by the demonstrator. Second, the EVD explicitly takes into account the initial state distribution. Thus, the EVD measures the generalization error of an evaluation policy by evaluating the expected return over all states with support under S_0, even if demonstrations have only been sampled from a small number of possible initial states.

The downside of using EVD is that it is more computationally intensive than the feature count bound. Computing EVD requires that we calculate V_R^* and $V_R^{\pi_{\text{eval}}}$ for each sample

R. Note however, that computing the optimal policy π^* is already required for Bayesian IRL (see Section 3.3). Thus, we can simply save V_R^* during MCMC. Furthermore, given the assumption of a linear reward function, $R(s) = w^T\phi(s)$, we have $V_R^{\pi_{\text{eval}}} = w^T\Phi_{\pi_{\text{eval}}}$. Thus, we only need to solve for $\Phi_{\pi_{\text{eval}}}$ once via standard policy evaluation (Sutton and Barto, 1998) and then $V_R^{\pi_{\text{eval}}}$ can be solved via a simple dot product for every sample $R \sim P(R|D)$. Thus, our approach leverages the computation already performed during Bayesian IRL to allow us to compute high-confidence policy evaluation bounds with minimal extra overhead.

To bound the α-quantile worst-case EVD(π_{eval}, R^*) we use samples from the posterior $P(R|D)$. Thus, we seek to calculate $\nu_\alpha(Z)$ where $Z = EVD(\pi_{\text{eval}}, R)$ for $R \sim P(R|D)$. As motivated earlier, we assume $\|w\|_1 = 1$. Thus, to find $P(R|D)$ we use a modified version of the Bayesian IRL Policy Walk Algorithm (Ramachandran and Amir, 2007) that ensures that our proposal samples of w during MCMC stay on the L1-norm unit ball. Details are given in Appendix A.2. Using MCMC, we generate a sequence of sampled rewards $\mathcal{R} = \{R : R \sim P(R|D)\}$ from the posterior distribution over reward functions given the demonstrations. For each sample $R_i \in R$ we then calculate

$$Z_i = \text{EVD}(\pi_{\text{eval}}, R_i) = V_{R_i}^* - V_{R_i}^{\pi_{\text{eval}}} \tag{4.18}$$

giving us samples from the posterior distribution over expected value differences.

Given n samples of X, we can obtain a point estimate of α-VaR by sorting the samples of X in ascending order to obtain the order statistics Z, and then take the α-quantile. This gives us Z_k, where $k = \lceil \alpha n \rceil$, as an estimate of the α-VaR. However, this does not take into account the number of samples or our confidence in this point estimate.

Because we are interested in high-confidence bounds, instead of using a point estimate, we compute a single-sided $(1 - \delta)$ confidence upper bound on the α-VaR. By definition, we have that $P(X_i < \nu_\alpha(X)) = \alpha$ for any sample X_i. Thus, for any order statistic Z_j, we can calculate the probability that the α-VaR is less than Z_j using the binomial cu-

Algorithm 1 $(1-\delta)$-Confidence Bound on the α-Value-at-Risk

input: MDP\R, π_{eval}, D, β, α, δ, n
$\mathcal{R} \leftarrow$ **Bayesian IRL**(MDP\R, D, β, n) ▷ sample n times from $P(R|D)$
for $R_i \in \mathcal{R}$ **do**
    $X_i = V^*_{R_i} - V^{\pi_{\text{eval}}}_{R_i}$ ▷ compute policy loss
$Z = \text{sort}(X)$ ▷ sort into ascending order statistics
$k = F^{-1}_{\text{bin}}(1-\delta, n, \alpha)$ ▷ index of $(1-\delta)$-confidence upper bound on α-VaR(X)
return Z_k

mulative distribution function (CDF), F_{bin}:

$$P(\nu_\alpha(X) < Z_j) = F_{\text{bin}}(j-1; n, \alpha) \tag{4.19}$$

$$= \sum_{i=0}^{j-1} \binom{n}{i} \alpha^i (1-\alpha)^{N-i} \tag{4.20}$$

This is because for Z_j to be larger than the 100α percentile value of X, we must have that this percentile value, namely $\nu_\alpha(X)$, is greater than no more than $j-1$ of the samples. This probability is given by the CDF of the binomial distribution, $F_{\text{bin}}(j-1; n, \alpha)$, which gives the probability of getting $j-1$ or fewer successes in n trials. In this case, a success is when a sample of X_i is less than $\nu_\alpha(X)$, and thus the probability of success is $P(X_i < \nu_\alpha(X)) = \alpha$ by definition of $\nu_\alpha(Z)$. Thus, using the inverse CDF of the binomial distribution, the order statistic Z_k, where $k = F^{-1}_{\text{bin}}(1-\delta, n, \alpha)$, forms a $(1-\delta)$ confidence bound on $\nu_\alpha(X)$.[2]

Our full general approach, applicable to any MDP\R, is summarized in Algorithm 1. The optimized version that assumes a linear reward function is summarized in Algorithm 2. The algorithm has four hyperparameters: β represents the temperature parameter in the Boltzman rationality model of the demonstrator, α defines the risk-sensitivity, $(1-\delta)$ represents the desired confidence level on the estimate of the α-VaR, and n is the number of reward functions to sample from the posterior using MCMC.

[2]Most standard scientific computing packages have efficient implementations of the inverse binomial distribution. For example, in Python this can efficiently computed using the SciPy package, e.g., `scipy.stats.binom.ppf`$(1-\delta, n, \alpha)$.

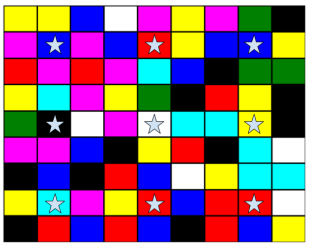
Algorithm 2 $(1-\delta)$-Confidence Bound on the α-Value-at-Risk for Linear Reward Function

input: MDP\R, π_{eval}, D, β, α, δ, n
$\Phi_{\pi_{\text{demo}}} \leftarrow$ **PolicyEvaluation**(π_{demo}, MDP\R) ▷ Compute expected feature counts
$\mathcal{W} \leftarrow$ **BayesianIRL**(MDP\R, D, β, n) ▷ sample n pairs (w, V_w^*) from $P(w|D)$
for $(w_i, V_{w_i}^*) \in \mathcal{W}$ **do**
    $X_i = V_{w_i}^* - w_i^T \Phi_{\pi_{\text{demo}}}$ ▷ compute policy loss
$Z = \text{sort}(X)$ ▷ sort into ascending order statistics
$k = F_{\text{bin}}^{-1}(1-\delta, n, \alpha)$ ▷ index of $(1-\delta)$-confidence upper bound on α-VaR(X)
return Z_k

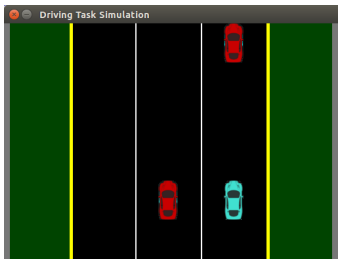
The main advantages of our approach are as follows: (1) our proposed bound takes full advantage of all of the information contained in the transition dynamics and demonstrations to focus on reward functions that are likely given the demonstrations, (2) it does not require optimal demonstrations, (3) it inherits from Bayesian IRL the ability to work with partial demonstrations, even disjoint state-action pairs, and (4) it allows for domain knowledge in the form of a prior.

4.5 Empirical Results

For our proposed confidence bound to be useful, it needs to meet several criteria: (1) the upper bound should be accurate with high-confidence (rarely underestimating the true expected value difference), (2) the bound should be tighter than the worst-case bound derived above, and (3) the previous two criteria should be true even when given a small number of demonstrations. We use both a standard grid world navigation task (Abbeel and Ng, 2004; Ramachandran and Amir, 2007; Choi and Kim, 2011) and a simulated driving task (Abbeel and Ng, 2004; Syed and Schapire, 2007; Cohn et al., 2011) to validate that our proposed bound satisfies these criteria. Examples of these tasks are shown in Figure 4.1. We compare our high-confidence α-VaR bound with the worst-case feature count bound (WFCB) defined in Equation 4.16. All results for α-VaR bounds are reported as 95% confidence



(a) Gridworld navigation



(b) Driving simulation

Figure 4.1: (a) Example of random grid world navigation task with colors representing random features and initial states denoted by stars. (b) Snapshot of driving simulation. Agent must learn to safely drive the blue car through traffic.

bounds ($\delta = 0.05$). [3]

We first empirically evaluate our approach on a suite of 9x9 grid world navigation tasks where the cost of traveling on different terrains is unknown and must be inferred from demonstrations. The available actions are up, down, left and right. Transitions are noisy with an 70% chance of moving in the desired direction and 30% chance of going in one of the directions perpendicular to the chosen direction. There are 8 binary features with one feature active per grid cell. To show that our results are not an artifact of a specific reward function, we evaluate our method over many random grid worlds, each with a randomly chosen ground truth reward. We use $\gamma = 0.9$ and an initial state distribution S_0 that is uniform over 9 different states spread across the grid as shown in Figure 4.1a. When generating M demonstrations we select the initial states in a round-robin fashion from the support of S_0. However, when measuring accuracy and bound errors, we compare with the true expected value difference over the full initial state distribution.

4.5.1 Infinite Horizon Grid Navigation

Our first task is an infinite horizon grid world navigation task with no terminal states. To evaluate different bounding methods we generated 200 random 9x9 worlds with random

[3] Code and instructions to reproduce the results in this chapter are available at https://github.com/dsbrown1331/aaai-2018-code

features each grid cell. For each world we generated a random feature weight vector w from the L1-unit norm ball. To generate demonstrations we solve the MDP using the random ground truth reward to find the optimal policy and use this policy to generate trajectories of length 100. We set the evaluation policy to be the optimal policy under the MAP reward function found using Bayesian IRL. Because the demonstrations in this experiment are perfect, we set the Bayesian IRL confidence parameter to a large value ($\beta = 100$).

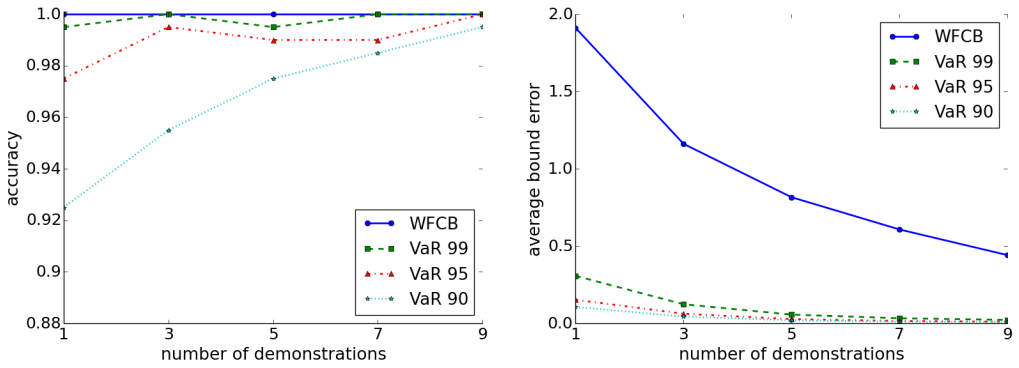
Figure 4.2a shows the accuracy of each bound where WFCB is the worst-case feature count bound, and VaR X is the X/100 quantile Value at Risk bound. The accuracy is the proportion of trials where the upper bound is greater than the ground truth expected value difference over the 200 random grid worlds. As expected, the WFCB always gives an upper bound on the true performance difference between the optimal policy and the evaluation policy. The bounds on α-VaR are also highly accurate. Because always predicting a high upper bound will result in high accuracy, we also measured the tightness of the the upper bounds. Figure 4.2b shows the average bound error over the 200 random navigation tasks. We define the bound error for an upper bound b as

$$\text{error}(b) = b - \text{EVD}(\pi_{\text{eval}}, R^*) \tag{4.21}$$

where R^* is the generated ground truth reward. We see that the bounds on the α-VaR are much tighter than the worst-case feature count bound, converging after only a small number of demonstrations.

4.5.2 Noisy Demonstrations

As mentioned previously, Bayesian IRL uses a confidence parameter, β, that represents the optimality of the demonstrations. When $\beta = 0$, the demonstrations are assumed to come from a completely random policy, and $\beta = \infty$ means that the demonstrations come from a perfectly optimal policy. Prior work used values of β between 25 and 500 when demonstrations are generated from an expert policy (Lopes et al., 2009; Cohn et al., 2011;



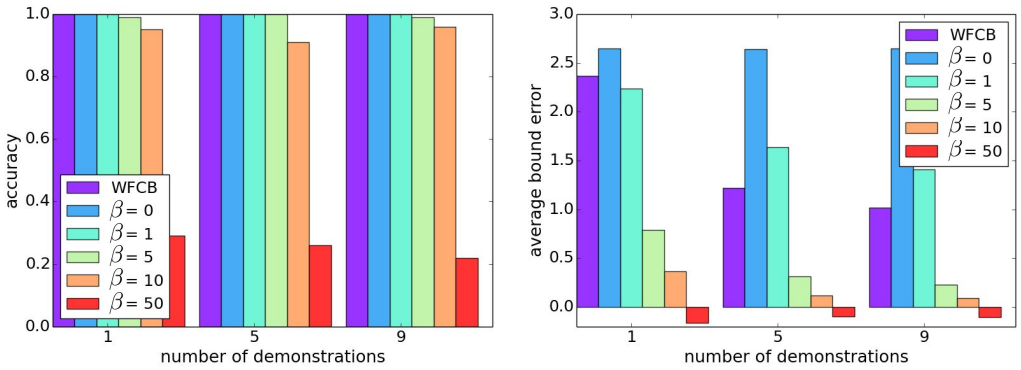
(a) Accuracy

(b) Average Bound Error

Figure 4.2: Results for infinite horizon grid navigation task. Accuracy and average error for bounds based on feature counts (WFCB) compared with 99, 95, and 90 percentiles for the VaR bound. Accuracy and averages are computed over 200 replicates

Michini and How, 2012b). To investigate the effect of β on our bound we generated noisy demonstrations where at step there is an 80% chance of taking an optimal action and a 20% chance of taking a random action. The resulting accuracy and bound error for several choices of β are shown in Figure 4.3.

Adjusting β for noisy demonstrations has a clear effect on the accuracy and bound error. The bound error (Equation 4.21) decreases as β increases, meaning the bounds become tighter; however, when $\beta = 50$ the VaR bounds often underestimate the true expected value difference between the expert's policy and the evaluation policy, resulting in error$(b) < 0$ and lower accuracy. We see that values of β in the range $(1, 10]$ result in highly accuracy bounds that are tighter than the worst-case feature count bound. However, for $\beta = 50$, we see that Bayesian IRL overfits to the noise in the demonstrations by assuming that the demonstrations are optimal. Tuning the confidence parameter, β, for a particular demonstrator and task is left for future work.



(a) Accuracy

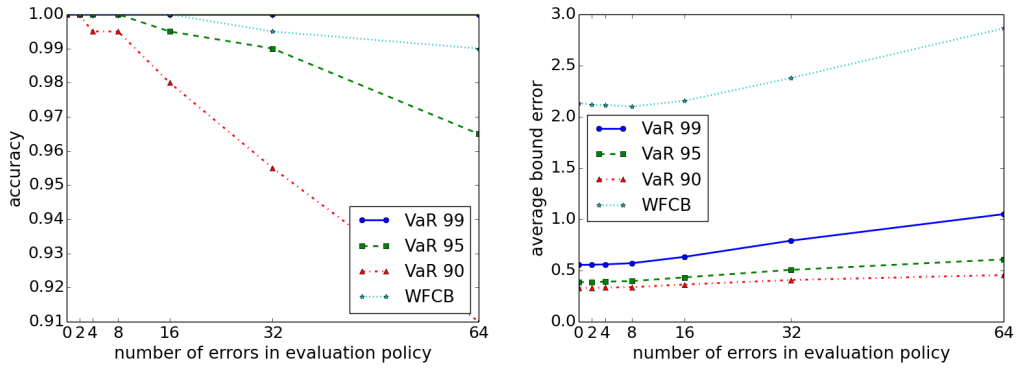
(b) Average Bound Error

Figure 4.3: Sensitivity to the confidence β for noisy demonstrations in the grid navigation task. The demonstrator has a 20% chance of taking a random action in each state. Accuracy and average error for bounds based on feature counts (WFCB) compared with 0.95-VaR bound. Accuracy and averages are computed over 200 replicates.

4.5.3 Sensitivity to Evaluation Policy

In the previous examples we have used the MAP reward obtained from Bayesian IRL to create the evaluation policy; however, unlike previous theoretical confidence bounds, our method is applicable to any evaluation policy. We investigated the sensitivity of our bound over a range of different evaluation policies and found that the VaR bounds consistently outperforms the baseline WFCB, providing bounds that are often four times tighter while maintaining high accuracy.

In this section we investigate how the bound on VaR is affected by the choice of evaluation policy. We ran an experiment where we varied the optimality of the evaluation policy. As in our previous experiments, used a 9x9 grid world and we generated 200 MDPs with random features and feature weights for evaluation. The evaluation policy was chosen by taking the optimal policy obtained from the ground truth reward and selecting X states at random without replacement and changing the policy at those X states so that it takes a non-optimal action. All demonstrations were optimal so we computed the VaR bounds using $\beta = 100$.

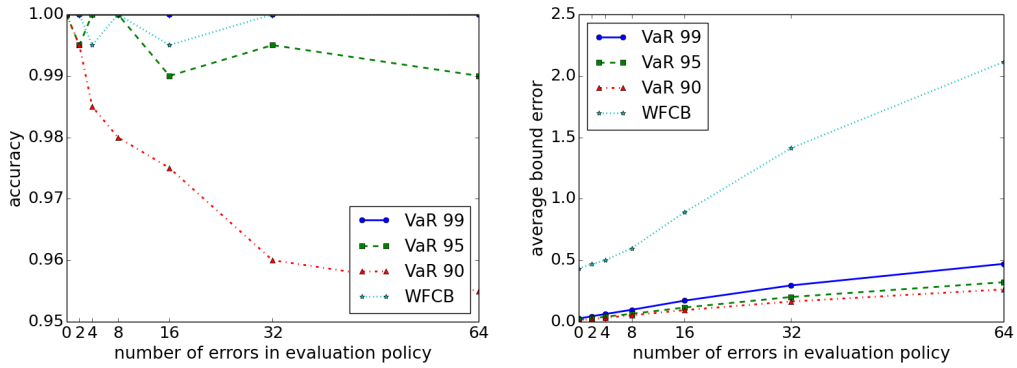


(a) Accuracy after one demonstration (b) Average bound error after one demonstration

Figure 4.4: Sensitivity for bounding the performance of a range of evaluation policies given 1 optimal demonstration. Results are averaged over 200 grid navigation task with no terminal states. Accuracy and average error for WFCB bounds versus bounds on the 0.99-, 0.95-, and 0.90-VaR.

The results for $X = 0, 2, 4, 8, 16, 32, 64$ after one demonstration are shown in Figure 4.4 and the results after nine demonstrations are shown in Figure 4.5. When the evaluation matches the optimal policy under the true reward ($X = 0$) all bound methods always gave true upper bounds on the EVD. When only one demonstration is given, there is a large bound error for all methods, with WFCB giving an error bound 4 times higher than the worst VaR bound error. As X is increased, the evaluation policy becomes more dissimilar to the optimal policy. This results in a drop in accuracy for all bounds except for the extremely conservative 0.99-VaR bound. When 9 demonstrations are given, the VaR bounds are much tighter with almost zero error for evaluation bounds close to optimal. The accuracy tends to drop as the number of errors increases, but still remains above 95% even for a policy that differs from the optimal policy in over 75% of the states.

To demonstrate the ability of our method to work with evaluation policies derived from other IRL algorithms, and to compare against existing high-confidence bounds for IRL, we used the Projection algorithm proposed by Abbeel and Ng (2004) as an evaluation policy. Abbeel and Ng provide high-confidence bounds on the number of demonstrations



(a) Accuracy after nine demonstrations

(b) Bound error after nine demonstrations

Figure 4.5: Sensitivity for bounding the performance of a range of evaluation policies given 9 optimal demonstrations. Results are averaged over 200 grid navigation task with no terminal states. Accuracy and average error for WFCB bounds versus bounds on the 0.99-, 0.95-, and 0.90-VaR.

needed for their algorithm to guarantee performance within ϵ of the demonstrator. A tighter sample bound for feature count-based methods was later derived by (Syed and Schapire, 2007) that also holds for the Projection algorithm. We summarize their result as the following theorem.

Theorem 1. *(Syed and Schapire, 2007) To obtain a policy $\hat{\pi}$ such that with probability $(1-\delta)$*

$$\epsilon \geq |V^{\hat{\pi}}(R^*) - V^{\pi^*}(R^*)| \tag{4.22}$$

it suffices to have

$$m \geq \frac{2}{(\frac{\epsilon}{3}(1-\gamma))^2} \log \frac{2k}{\delta}. \tag{4.23}$$

In the following corollary we invert the bound of Syed and Schapire by solving for ϵ to obtain a $(1-\delta)$ confidence bound on the expected value difference given a fixed number of demonstrations.

Corollary 1. *Given a confidence level δ, and m demonstrations, with probability $(1-\delta)$*

	Number of demonstrations					Avg. Accuracy
	1	5	9	$\cdots$	23,146	
0.95-VaR Bound	**0.9372**	**0.2532**	**0.1328**		-	0.98
0.99-VaR Bound	1.1428	0.2937	0.1535		-	1.0
(Syed and Schapire, 2007)	142.59	63.77	47.53		0.9372	1.0

Table 4.1: Comparison of 95% confidence α-VaR bounds with a 95% confidence Hoeffding bound (Syed and Schapire, 2007). Both bounds use the Projection algorithm (Abbeel and Ng, 2004) to obtain the evaluation policy. Results are averaged over 200 random navigation tasks.

we have that $|V^{\pi^*}(R^*) - V^{\hat{\pi}}(R^*)| \leq \epsilon$*, where*

$$\epsilon \leq \frac{3}{1-\gamma}\sqrt{\frac{2}{m}\log\frac{2k}{\delta}} \tag{4.24}$$

where k *is the number of features and* γ *is the discount factor of the underlying MDP.*

We then repeated the infinite horizon grid navigation experiment described above, using the policy found by the Projection algorithm as our evaluation policy. We compare the average bound error for our proposed VaR bounds with the Syed and Schapire error bound for the Projection algorithm in Table 4.1. Our empirical VaR bounds are two to three orders of magnitude tighter than the Hoeffding style bound which theoretically requires 23,146 demonstrations to guarantee the true EVD is within the 0.95-VaR bound found by our method using only a single demonstration.

4.5.4 High-Confidence Policy Selection for a Simulated Driving Task

We now provide an example that more closely matches a real-world learning from demonstration task. Rather than evaluate our method on an ad hoc "true" reward function, we examine how the VaR bound can be used to rank and select an appropriate policy from a set of existing policies. For this task we designed a driving simulator based on previous benchmarks (Abbeel and Ng, 2004; Cohn et al., 2011). Figure 4.1b shows a snapshot of

the simulator. The agent (blue) is in charge of driving safely down a highway and has three actions: switch lanes left, switch lanes right, or stay in current lane. The agent is traveling faster than traffic and must change lanes to avoid other cars which randomly appear at the top of the screen. There are three highway lanes where the car is supposed to drive, but it can also drive off-road on the right or left of the highway.

The state space is made up of 12 binary features: 5 features for each of the possible lanes, including the off-road lanes, 3 features telling the agent whether it is currently in collision, tailgating, or trailing another car, and 2 features for each adjacent lane, indicating whether the car will be in collision or tailgating if the car changes lanes. The reward is assumed to be a linear combination of features, $R(s) = w^T\phi(s)$, where $\phi(s)$ is a 6-dimensional binary feature vector that indicates the agent's current lane and whether it is in collision with another car. The discount factor, γ, was set to 0.9.

The goal of this experiment is to evaluate the ability of our probabilistic performance bound to correctly rank different policies, given a single demonstration of safe driving. We constructed three different evaluation policies: (1) **right-safe**: a policy that avoids hitting cars and driving off-road, but prefers driving on the right lane of the highway, (2) **on-road**: a policy that avoids driving off-road, but pays no attention to other cars, and changes lanes randomly (3) **nasty**: a policy that avoids going off-road, but actively tries to hit cars. We then generated a single demonstration of collision-free driving, consisting of 100 consecutive state-action pairs. The demonstration changed lanes randomly while avoiding collisions and avoiding driving off-road. The evaluation policies and demonstration were created using Q-learning and hand-crafted reward functions that resulted in the desired behaviors.[4]

Because the driving task is model-free we used Q-learning to calculate the Q-values used in the likelihood calculations of Bayesian IRL. We then calculated a 95% confidence bound on the 0.95-VaR for each evaluation policy. We also computed the worst-case feature

[4]Videos of these behaviors are shown in the code documentation available at https://github.com/dsbrown1331/aaai-2018-code

		Ranking (EVD upper bound)		
π_{eval}	Collisions	True	WFCB	0.95-VaR
right-safe	0	**1**	3 (5.51)	**1** (0.85)
on-road	13.65	**2**	1 (1.93)	**2** (1.09)
nasty	42.75	**3**	2 (4.11)	**3** (2.44)

Table 4.2: Policy rankings based on upper bounds on policy loss for three different evaluation policies in the driving domain when given a single demonstration of safe driving. Results are averaged over 20 replicates.

count bounds for comparison. The results are shown in Table 4.2.

The VaR bound uses the demonstration to focus on reward functions that are likely given the demonstrated state-action pairs. This results in correctly ranking the evaluation policies. The worst-case feature count bound ignores likelihood and assumes a worst-case reward function that penalizes the largest discrepancy between the empirical feature counts of the demonstration and the expected feature counts of the evaluation policies. Because the collision feature is less frequently active than the lane features, both **on-road** and **nasty** appear safer than **right-safe** because their average state-occupancies more closely align with the state-occupancies of the demonstration.

4.5.5 High-confidence policy improvement

The previous section showed how we can use risk-sensitive policy evaluation to choose between multiple evaluation policies. We now take this a step further and give an example that uses risk-sensitive policy evaluation to iteratively reduce the VaR of a policy learned from demonstrations.

To highlight the potential of safe policy improvement, we consider the simple navigation task shown in Figure 4.6. The task has a single terminal in the center and two reward features (white and red). The agent is given a single demonstration from one starting state and must generalize this demonstration to a second starting state (both marked with circles). Note that the demonstration shows that the red feature is less desirable than the white

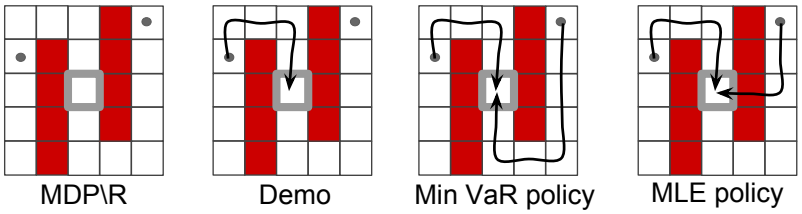


Figure 4.6: Given one demonstration, optimizing the VaR bound results in a risk-aware policy that hedges against the red cells being much worse than the white. The maximum likelihood reward assumes that red is only marginally worse than white.

feature, but the true magnitudes of the feature weights are left uncertain.

We implemented a simple risk-sensitive policy improvement hill climbing algorithm. We initialized the hill climbing algorithm with the maximum likelihood policy found using Bayesian IRL with a uniform prior. For each step of the hill-climbing algorithm, we examined the impact on the 0.99-VaR of changing the action taken by the policy in a single state, and chose the change that resulted in the largest decrease in 0.99-VaR over all single state changes. We continued this process until no reductions in the 0.99-VaR could be found. The resulting risk-aware policy seeks to minimize the 0.99-VaR by avoiding the red feature, whereas the maximum likelihood reward leads to a less conservative policy, resulting in a higher potential risk. The learned policies are shown in Figure 4.6. In Chapter 7 we introduce a more elegant and efficient robust policy optimization algorithm that avoids the hill-climbing approach presented here and instead directly optimizes a policy such that it balances risk and return.

4.6 Summary

In this chapter we formalized and addressed the problem of safe imitation learning via high-confidence policy evaluation under an unknown reward function. To our knowledge, this work provides the first general framework for obtaining practical high-confidence bounds on the performance difference between an evaluation policy and the optimal policy for a

demonstrator's true unknown reward. We also gave examples of how our high-confidence performance bound can be used to perform risk-aware policy selection and risk-aware policy improvement. Our proposed algorithms are evaluated on a standard grid navigation task and driving simulation.

Our results demonstrate that our proposed bound is a significant improvement over a baseline based on feature counts—providing accurate, tight bounds even for small numbers of demonstrations. Additionally, our empirical results show orders of magnitude improvement in sample efficiency over competing confidence bounds (Abbeel and Ng, 2004; Syed and Schapire, 2007). As a result, this is the first approach that allows agents that learn from demonstrations to express confidence in the performance of their learned policy, based on limited demonstration data. We believe the techniques proposed in this chapter provide a starting point for developing autonomous agents that can safely and efficiently learn from human demonstrations in risk-sensitive, real-world environments.

The method proposed in this section provides a high-confidence upper bound on the policy loss of any evaluation policy π_{eval}. However, what if a learning agent's policy has a high value-at-risk? In this case, there is a chance that the robot's policy may significantly deviate from the intent of the demonstrator. In Chapter 8 we address this issue by presenting an active learning algorithm that uses risk-aware active queries to perform robust policy improvement. In Chapter 7 we extend our work on robust policy optimization described in Section 4.5.5 and present a more principled and efficient algorithm that directly performs robust policy optimization with respect to the Conditional Value at Risk of a policy under a posterior distribution over reward functions.

One of the main drawbacks of our proposed framework is that it requires running MCMC, which requires solving for $Q^*_R(s,a)$ and V^*_R in order to calculate the Bayesian IRL likelihood and evaluate the likelihood ratio of each proposal R. Furthermore, our results in Section 4.5.2 demonstrate that unless β is tuned correctly the performance of our method degrades as demonstrations become more suboptimal. In the next chapter we present re-

ward learning approaches that utilize preference rankings over demonstrations to remove the need for an MDP solver in the inner-loop and enable learning from highly suboptimal demonstrations. Finally, in Chapter 6 we demonstrate how ranked demonstrations enable efficient solutions to the high-confidence policy evaluation problem for imitation learning in high-dimensional control tasks.

Chapter 5

Computationally Efficient Reward Learning from Suboptimal Demonstrations

The previous chapter formalized and presented a sample efficient solution to the problem of high-confidence policy evaluation for imitation learning (HCPE-IL). However, while sample efficient in terms of the number of demonstrations, the method presented in Chapter 4 requires an MDP solver in the inner-loop, making it intractable for complex imitation learning problems where solving for an optimal policy is difficult. Furthermore, our previous results assumed that the demonstrator was near-optimal, which is not always true. Thus, if we want to compute high-confidence performance bounds for complex imitation learning tasks, we first need to address some of the limitations in standard imitation learning approaches. This chapter focuses on improving the efficiency of reward function inference. Later, in Chapter 6 we will combine results from the current chapter with results from Chapter 4 to propose a method for efficient high-confidence policy evaluation that scales to visual imitation learning tasks. This chapter presents contributions 3, 4, and 5 of this dissertation: theoretical results for better-than-demonstrator imitation learning and

preference-based inverse reinforcement learning; a computationally efficient algorithm for reward learning from suboptimal, ranked observations that scales to high-dimensional tasks and can outperform the demonstrator; and computationally efficient algorithms for learning to extrapolate intention from unlabeled suboptimal demonstrations.[5]

While imitation learning is a popular paradigm to teach robots and other autonomous agents to perform complex tasks simply by showing examples of how to perform the task, one of the drawbacks of standard imitation learning is that it typically optimizes policies whose performance is upper-bounded by the performance of the demonstrator. While it is possible to learn policies that perform better than a demonstrator, existing methods either require access to a hand-crafted reward function (Hester et al., 2018) or a human supervisor who acts as a reward or value function during policy learning (Christiano et al., 2017; Brown et al., 2019b).

Imitation learning approaches based on on inverse reinforcement learning are further limited because they usually have to solve a complex reinforcement learning step in the inner-loop. IRL algorithms typically either require fully solving an MDP solver in the inner-loop (Abbeel and Ng, 2004; Ziebart et al., 2008; Ramachandran and Amir, 2007) or partially solving an MDP in the inner-loop which requires collecting large amounts of trajectory data during reward function inference and usually requires expensive rollouts in the real world or a high-fidelity model or simulator (Finn et al., 2016; Ho and Ermon, 2016; Fu et al., 2017).

In this chapter we seek to remedy both of these problems via reward learning from pre-ranked demonstrations. We first present theoretical results for when better-than-demonstrator performance is possible in an inverse reinforcement learning (IRL) setting (Abbeel and Ng, 2004), where the goal is to recover a reward function from demonstrations. We then present theoretical results demonstrating that rankings (or alternatively, pair-

[5]This chapter contains work that was done in collaboration with Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum and was previously published at ICML 2019 (Brown et al., 2019b) and CoRL 2019 (Brown et al., 2019a).

wise preferences) over demonstrations can enable better-than-demonstrator performance by reducing error and ambiguity in the learned reward function. Next, we present a computationally efficient method for inverse reinforcement learning via pre-ranked demonstrations. Our approach allows an imitation learning agent to infer the reward function of a demonstrator via preferences which removes the need for an MDP solver or any inference time data collection.

Finally, we present work that leverages the benefits of reward learning via ranked demonstrations in a way that does not require human rankings. Requiring a demonstrator to rank demonstrations can be tedious and error prone, and precludes learning from prerecorded, unranked demonstrations, or learning from demonstrations of similar quality that are difficult to rank. Thus, we investigate whether it is possible to generate a set of ranked demonstrations, in order to surpass the performance of a demonstrator, without requiring supervised preference labels or reward information. We evaluate two methods for automatically ranking demonstrations: (1) watching someone improve at a task over time and (2) injecting noise into a cloned policy.

5.1 Better-than-Demonstrator Performance: Theory

We model the environment as a Markov decision process or MDP (see Section 3.1). Given a policy and an MDP, we will denote the expected discounted return of the policy as given by $J(\pi|R^*) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)]$. Similarly, the return of a trajectory consisting of states and actions, $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$, is denoted by $J(\tau|R^*) = \sum_{t=0}^{T} \gamma^t R^*(s_t)$.

We assume that we have no access to the true reward function of the MDP. Instead, we are given a set of m demonstrations $\mathcal{D} = \{\tau_1, \ldots \tau_m\}$, where each demonstrated trajectory is can be a sequence of states and actions, $\tau_i = (s_0, a_0, s_1, a_1, \ldots)$ or, in the case of imitation from observation, a sequence of states, $\tau_i = (s_0, s_1, s_2, \ldots)$. We assume that the demonstrator is attempting (possibly unsuccessfully) to follow a policy that optimizes the true reward function R^*. Given the demonstrations $\mathcal{D}$, we wish to find a

policy $\hat{\pi}$ that can extrapolate beyond the performance of the demonstrator. We say a policy $\hat{\pi}$ can extrapolate beyond of the performance of the demonstrator if it achieves a larger expected return than the demonstrations, when evaluated under the true reward function R^*, i.e., $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$, where $J(\mathcal{D}|R^*) = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}} J(\tau|R^*)$. Similarly, we say that a learned policy $\hat{\pi}$ extrapolates beyond the performance of the best demonstration if $J(\hat{\pi}|R^*) > \max_{\tau\in\mathcal{D}} J(\tau|R^*)$.

5.1.1 Extrapolating Beyond a Demonstrator

We first provide a sufficient condition under which it is possible to achieve better-than-demonstrator performance in an inverse reinforcement learning (IRL) setting, where the goal is to recover the demonstrator's reward function which is then used to optimize a policy (Arora and Doshi, 2018). We consider a learner that approximates the reward function of the demonstrator with a linear combination of features: $R(s) = w^T\phi(s)$.[6] These can be arbitrarily complex features, such as the activations of a deep neural network. Under the assumption of a linear reward function, we have

$$J(\pi|R) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\right] = w^T\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)\right] = w^T\Phi_\pi, \tag{5.1}$$

where Φ_π are the expected discounted feature counts that result from following policy π.

Theorem 2. *If the estimated reward function is $\hat{R}(s) = w^T\phi(s)$, the true reward function is $R^*(s) = \hat{R}(s) + \epsilon(s)$ for some error function $\epsilon : \mathcal{S} \to \mathbb{R}$, and $\|w\|_1 \leq 1$, then extrapolation beyond the demonstrator, i.e., $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$, is guaranteed if :*

$$J(\pi^*_{R^*}|R^*) - J(\mathcal{D}|R^*) > \epsilon_\Phi + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{5.2}$$

[6] Our results also hold for reward functions of the form $R(s,a) = w^T\phi(s,a)$.

where $\pi^*_{R^*}$ *is the optimal policy under* R^*, $\epsilon_\Phi = \|\Phi_{\pi^*_{R^*}} - \Phi_{\hat{\pi}}\|_\infty$ *and*

$$\|\epsilon\|_\infty = \sup\{\,|\epsilon(s)| : s \in \mathcal{S}\,\}. \tag{5.3}$$

Proof. See Appendix B.1. □

Intuitively, extrapolation depends on the demonstrator being sufficiently suboptimal, the error in the learned reward function being sufficiently small, and the state occupancy of the imitation policy, $\hat{\pi}$, being sufficiently close to $\pi^*_{R^*}$. If we can perfectly recover the reward function, then reinforcement learning can be used to ensure that ϵ_Φ is small. Thus, we focus on improving the accuracy of the learned reward function via automatically-ranked demonstrations. The learned reward function can then be optimized with any reinforcement learning algorithm (Sutton and Barto, 1998).

5.1.2 Extrapolation via ranked demonstrations

The previous results demonstrate that in order to extrapolate beyond a suboptimal demonstrator, it is sufficient to have small reward approximation error and a good policy optimization algorithm. However, the following proposition, adapted from (Castro et al., 2019), shows that the reward function learned by standard IRL may be quite superficial and miss potentially important details, whereas enforcing a ranking over trajectories leads to a more accurate estimate of the true reward function.

Proposition 1. *There exist MDPs with true reward function* R^**, expert policy* π_E*, approximate reward function* $\hat{R}$*, and non-expert policies* π_1 *and* π_2*, such that*

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|R^*) \;\; \textit{and} \;\; J(\pi_1|R^*) \ll J(\pi_2|R^*) \tag{5.4}$$

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \;\; \textit{and} \;\; J(\pi_1|\hat{R}) = J(\pi_2|\hat{R}). \tag{5.5}$$

However, enforcing a preference ranking over trajectories, $\tau^* \succ \tau_2 \succ \tau_1$*, where* $\tau^* \sim \pi^*$*,*

$\tau_2 \sim \pi_2$, and $\tau_1 \sim \pi_1$, results in a learned reward function $\hat{R}$, such that

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \;\; \textit{and} \;\; J(\pi_1|\hat{R}) < J(\pi_2|\hat{R}). \tag{5.6}$$

Proof. See Appendix B.2. □

Proposition 1 proves the existence of MDPs where an approximation of the true reward leads to an optimal policy, yet the learned reward reveals little about the underlying reward structure of the MDP. This is problematic for several reasons. The first problem is that if the learned reward function is drastically different than the true reward, this can lead to poor generalization. Another problem is that many learning from demonstration methods are motivated by providing *non-experts* the ability to program by example. Some non-experts will be good at personally performing a task, but may struggle when giving kinesthetic demonstrations (Akgun et al., 2012) or teleoperating a robot (Chuck et al., 2017; Kent et al., 2017). Other non-experts may not be able to personally perform a task at a high level of performance due to lack of precision or timing, or due to physical limitations or impairment. Thus, the standard IRL approach of finding a reward function that maximizes the likelihood of the demonstrations may lead to an incorrect, superficial reward function that overfits to suboptimal user behavior in the demonstrations.

Indeed, it has been proven that it is impossible to recover the correct reward function without additional information beyond observations, regardless of whether the policy is optimal (Ng and Russell, 2000) or suboptimal (Armstrong and Mindermann, 2018). As demonstrated in Proposition 1, preference rankings can help to alleviate reward function ambiguity. If the true reward function is a linear combination of features, then the feasible region of all reward functions that make a policy optimal can be defined as an intersection of half-planes (Brown and Niekum, 2019b): $H_\pi = \bigcap_{\pi' \in \Pi} w^T(\Phi_\pi - \Phi_{\pi'}) \geq 0$. We define the *reward ambiguity*, $G(H_\pi)$, as the volume of this intersection of half-planes: $G(H_\pi) = \text{Volume}(H_\pi)$, where we assume without loss of generality that $\|w\| \leq 1$, to

ensure this volume is bounded. We prove that a total ranking over policies results in less reward ambiguity than performing IRL on the optimal policy.

Proposition 2. *Given a policy class* Π*, an optimal policy* $\pi^* \in \Pi$ *and a total ranking over* Π*, and true reward function* $R^*(s) = w^T\phi(s)$*, the reward ambiguity resulting from* π^* *is greater than or equal to the reward ambiguity of using a total ranking, i.e.,* $G(H^*_\pi) \geq G(H_{\text{ranked}})$.

Proof. See Appendix B.3. □

Learning a reward function that respects a set of strictly ranked demonstrations avoids some of the ill-posedness of IRL (Ng and Russell, 2000) by eliminating a constant, or all-zero reward function. Furthermore, ranked demonstrations provide explicit information about both what to do as well as what *not* to do in an environment and each pairwise preference over trajectories gives a half-space constraint on feasible reward functions. In Appendix B.4 we prove that sampling random half-space constraints results in an exponential decrease in reward function ambiguity.

Theorem 3. *To reduce the volume of* $\mathcal{J}$ *such that* $J_k = \epsilon$*, then it suffices to have* k *random half-space constraints, where*

$$k = \log_2 \frac{J_0}{\epsilon} \tag{5.7}$$

Proof. See Appendix B.4. □

Corollary 2. *To reduce reward function ambiguity by x% it suffices to have* $k = \log_2(1/(1 - x/100))$ *random half-space constraints over reward function weights.*

Proof. See Appendix B.4. □

In practice, sampling random half-space constraints on the ground-truth reward function is infeasible. One approach, often used in preference learning algorithms (Christiano et al., 2017; Sadigh et al., 2017) is to interactively ask for preference queries during

policy optimization. However, this can be burdensome for a human who may have to answer hundreds of preference queries over the space of several hours. Rather than assuming access to an oracle during policy optimization, we assume access to a set of pre-ranked demonstrations. In the next section, we consider the case where the demonstrations are explicitly ranked. In Section 5.3, we extend our results to cases where explicit preference queries are not available. As we will show in the next section, automatically-generating preferences over demonstrations also improves the efficiency of IRL by removing the need for an MDP solver in the inner-loop and turning IRL into a supervised learning problem.

5.2 Trajectory-Ranked Reward Extrapolation

It can be difficult, even for experts, to design reward functions and objectives that lead to desired behaviors when designing autonomous agents (Ng et al., 1999; Amodei et al., 2016). When goals or rewards are difficult for a human to specify, inverse reinforcement learning (IRL) (Abbeel and Ng, 2004) techniques can be applied to infer the intrinsic reward function of a user from demonstrations. Unfortunately, high-quality demonstrations are difficult to provide for many tasks—for instance, consider a non-expert user attempting to give kinesthetic demonstrations of a household chore to a robot. Even for relative experts, tasks such as high-frequency stock trading or playing complex video games can be difficult to perform optimally. If a demonstrator is suboptimal, but their intentions can be ascertained, then a learning agent ought to be able to exceed the demonstrator's performance in principle. However, existing IRL algorithms fail to do this, typically searching for a reward function that makes the demonstrations appear near-optimal (Ramachandran and Amir, 2007; Ziebart et al., 2008; Finn et al., 2016; Henderson et al., 2018). Thus, when the demonstrator is suboptimal, IRL results in suboptimal behavior as well. Imitation learning approaches (Argall et al., 2009) that mimic behavior directly without reward inference, such as behavioral cloning (Torabi et al., 2018a), also suffer from the same shortcoming.

In this section, we address this shortcoming in current imitation learning methods

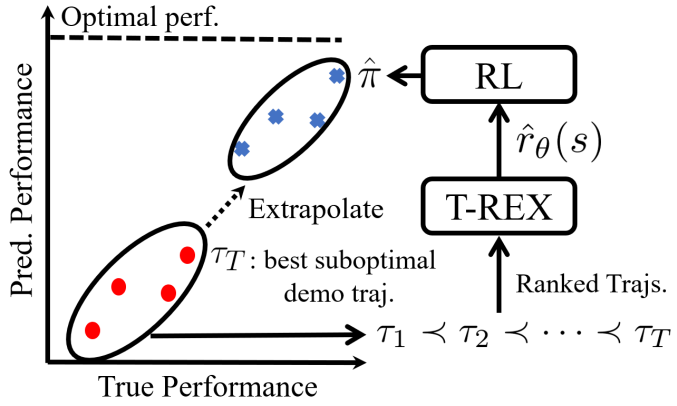


Figure 5.1: T-REX takes a sequence of ranked demonstrations and learns a reward function from these rankings that allows policy improvement over the demonstrator via reinforcement learning.

via preference rankings over demonstrations. In particular, we propose a novel reward inference algorithm, Trajectory-ranked Reward Extrapolation (T-REX) that utilizes a set of pre-ranked demonstrations to extrapolate a user's underlying intent beyond the best demonstration, even when all demonstrations are highly suboptimal. This, in turn, enables a reinforcement learning agent to exceed the performance of the demonstrator by learning to optimize this extrapolated reward function. Specifically, we use ranked demonstrations to learn a state-based reward function that assigns greater total return to higher-ranked trajectories. Thus, while standard inverse reinforcement learning approaches seek a reward function that *justifies* the demonstrations, we instead seek a reward function that *explains* the ranking over demonstrations, allowing for potentially better-than-demonstrator performance. T-REX is summarized in Figure 5.1.

Utilizing ranking in this way has several advantages. First, rather than imitating suboptimal demonstrations, it allows us to identify features that are correlated with rankings, in a manner that can be extrapolated beyond the demonstrations. Although the learned reward function could potentially overfit to the provided rankings, we demonstrate empirically that it extrapolates well, successfully predicting returns of trajectories that are signifi-

cantly better than any observed demonstration, likely due to the powerful regularizing effect of having many pairwise ranking constraints between trajectories. For example, the degenerate all-zero reward function (the agent always receives a reward of 0) makes any given set of demonstrations appear optimal. However, such a reward function is eliminated from consideration by any pair of (non-equally) ranked demonstrations. Second, when learning features directly from high-dimensional data, this regularizing effect can also help to prevent overfitting to the small fraction of state space visited by the demonstrator. By utilizing a set of suboptimal, but ranked demonstrations, we provide the neural network with diverse data from multiple areas of the state space, allowing an agent to better learn both what to do and what not to do in a variety of situations.

We evaluate T-REX on a variety of standard Atari and MuJoCo benchmark tasks. Our experiments show that T-REX can extrapolate well, achieving performance that is often more than twice as high as the best-performing demonstration, as well as outperforming state-of-the-art imitation learning algorithms. We also show that T-REX performs well even in the presence of significant ranking noise

5.2.1 Problem Definition

We assume access to a sequence of m ranked trajectories τ_t for $t = 1, \ldots, m$, where $\tau_i \prec \tau_j$ if $i < j$, we wish to find a parameterized reward function $\hat{r}_\theta$ that approximates the true reward function r that the demonstrator is attempting to optimize. Given $\hat{r}_\theta$, we then seek to optimize a policy $\hat{\pi}$ that can outperform the demonstrations.

Note that while our experiments in this section will typically involve a total ranking over demonstrations, our method works even with partial preference orderings. Our method only requires qualitative preferences. These preference labels can be obtained using many possible methods, such as a demonstrator giving pairwise preferences over trajectories, rating each demonstration on a likert scale, or simply providing a sequence of demonstrations and annotating whether each demonstration is better or worse than the previous demon-

stration. Note that even if the demonstrator provides a small set of relative scores to rank the demonstrations, this does not mean we have access to oracle scores during policy optimization. It is still necessary to use the preferences over demonstrations to infer a reward function which explains why some demonstration trajectories are scored better than others, which is what our proposed method does.

5.2.2 Algorithm

We now describe Trajectory-ranked Reward Extrapolation (T-REX), an algorithm for using ranked suboptimal demonstrations to extrapolate a user's underlying intent beyond the best demonstration. Given a sequence of m demonstrations ranked from worst to best, $\tau_1, \ldots, \tau_m$, T-REX has two steps: (1) reward inference and (2) policy optimization.

Given the ranked demonstrations, T-REX performs reward inference by approximating the reward at state s using a neural network, $\hat{r}_\theta(s)$, such that $\sum_{s\in\tau_i} \hat{r}_\theta(s) < \sum_{s\in\tau_j} \hat{r}_\theta(s)$ when $\tau_i \prec \tau_j$. The parameterized reward function $\hat{r}_\theta$ can be trained with ranked demonstrations using the generalized loss function:

$$\mathcal{L}(\theta) = \mathbf{E}_{\tau_i,\tau_j\sim\Pi}\Big[\xi\Big(\mathrm{P}\big(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j)\big), \tau_i \prec \tau_j\Big)\Big], \tag{5.8}$$

where Π is a distribution over demonstrations, ξ is a binary classification loss function, $\hat{J}$ is the return defined by a parameterized reward function $\hat{r}_\theta$, and $\prec$ is an indication of the preference between the demonstrated trajectories.

We represent the probability P as a softmax-normalized distribution and we instantiate ξ using a cross entropy loss:

$$\mathrm{P}\big(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j)\big) \approx \frac{\exp \sum_{s\in\tau_j} \hat{r}_\theta(s)}{\exp \sum_{s\in\tau_i} \hat{r}_\theta(s) + \exp \sum_{s\in\tau_j} \hat{r}_\theta(s)}, \tag{5.9}$$

$$\mathcal{L}(\theta) = -\sum_{\tau_i \prec \tau_j} \log \frac{\exp \sum_{s \in \tau_j} \hat{r}_\theta(s)}{\exp \sum_{s \in \tau_i} \hat{r}_\theta(s) + \exp \sum_{s \in \tau_j} \hat{r}_\theta(s)}. \tag{5.10}$$

This loss function trains a classifier that can predict whether one trajectory is preferable to another based on the predicted returns of each trajectory. This form of loss function follows from the classic Bradley-Terry and Luce-Shephard models of preferences (Bradley and Terry, 1952; Luce, 2012) and has been shown to be effective for training neural networks from preferences (Christiano et al., 2017; Ibarz et al., 2018).[7]

To increase the number of training examples, T-REX trains on partial trajectory pairs rather than full trajectory pairs. This results in noisy preference labels that are only weakly supervised; however, using data augmentation to obtain pairwise preferences over many partial trajectories allows T-REX to learn expressive neural network reward functions from only a small number of ranked demonstrations. During training we randomly select pairs of trajectories, τ_i and τ_j. We then randomly select partial trajectories $\tilde{\tau}_i$ and $\tilde{\tau}_j$ of length L. For each partial trajectory, we take each observation and perform a forward pass through the network $\hat{r}_\theta$ and sum the predicted rewards to compute the cumulative return. We then use the predicted cumulative returns as the logit values in the cross-entropy loss with the label corresponding to the higher ranked demonstration. Given the learned reward function $\hat{r}_\theta(s)$, T-REX then seeks to optimize a policy $\hat{\pi}$ with better-than-demonstrator performance through reinforcement learning using $\hat{r}_\theta$.[8]

5.2.3 MuJoCo Experiments and Results

We first evaluated our proposed method on three robotic locomotion tasks using the MuJoCo simulator (Todorov et al., 2012) within OpenAI Gym (Brockman et al., 2016), namely HalfCheetah, Hopper, and Ant. In all three tasks, the goal of the robot agent is to move for-

[7] Note that this can be implemented efficiently and robustly in most automatic differentiation software packages by simply using a binary cross entropy loss function where the logits are the predicted cumulative returns.

[8] Videos and code are available at https://github.com/hiwonjoon/ICML2019-TREX

ward as fast as possible without falling to the ground.

Demonstrations

To generate demonstrations, we trained a Proximal Policy Optimization (PPO) (Schulman et al., 2017) agent with the ground-truth reward for 500 training steps (64,000 simulation steps) and checkpointed its policy after every 5 training steps. For each checkpoint, we generated a trajectory of length 1,000. This provides us with different demonstrations of varying quality which are then ranked based on the ground truth returns. To evaluate the effect of different levels of suboptimality, we divided the trajectories into different overlapping stages. We used 3 stages for HalfCheetah and Hopper. For HalfCheetah, we used the worst 9, 12, and 24 trajectories, respectively. For Hopper, we used the worst 9, 12, and 18 trajectories. For Ant, we used two stages consisting of the worst 12 and 40 trajectories. We used the PPO implementation from OpenAI Baselines (Dhariwal et al., 2017) with the given default hyperparameters.

Experimental Setup

We trained the reward network using 5,000 random pairs of partial trajectories of length 50, with preference labels based on the trajectory rankings, not the ground-truth returns. To prevent overfitting, we represented the reward function using an ensemble of five deep neural networks, trained separately with different random pairs. Each network has 3 fully connected layers of 256 units with ReLU nonlinearities. We train the reward network using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 1e-4 and a minibatch size of 64 for 10,000 timesteps.

To evaluate the quality of our learned reward, we then trained a policy to maximize the inferred reward function via PPO. The outputs of each the five reward networks in our ensemble, $\hat{r}(s)$, are normalized by their standard deviation to compensate for any scale differences amongst the models. The reinforcement learning agent receives the average of

the ensemble as the reward, plus the control penalty used in OpenAI Gym (Brockman et al., 2016). This control penalty represents a standard safety prior over reward functions for robotics tasks, namely to minimize joint torques. We found that optimizing a policy based solely on this control penalty does not lead to forward locomotion, thus learning a reward function from demonstrations is still necessary.

Learned Policy Performance

We measured the performance of the policy learned by T-REX by measuring the forward distance traveled. We also compared against Behavior Cloning from Observations (BCO) (Torabi et al., 2018a), a state-of-the-art learning-from-observation method, and Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016), a state-of-the-art inverse reinforcement learning algorithm. BCO trains a policy via supervised learning, and has been shown to be competitive with state-of-the-art IRL (Ho and Ermon, 2016) on MuJoCo tasks without requiring action labels, making it one of the strongest baselines when learning from observations. We trained BCO using only the best demonstration among the available suboptimal demonstrations. We trained GAIL with all of the demonstrations. GAIL uses demonstrator actions, while T-REX and BCO do not.

We compared against three different levels of suboptimality (Stage 1, 2, and 3), corresponding to increasingly better demonstrations. The results are shown in Figure 5.2 (see Table 5.4 for full details). The policies learned by T-REX perform significantly better than the provided suboptimal trajectories in all the stages of HalfCheetah and Hopper. This provides evidence that T-REX can discover reward functions that extrapolate beyond the performance of the demonstrator. T-REX also outperforms BCO and GAIL on all tasks and stages except for Stage 2 for Hopper and Ant. BCO and GAIL usually fail to perform better than the average demonstration performance because they explicitly seek to imitate the demonstrator rather than infer the demonstrator's intention.

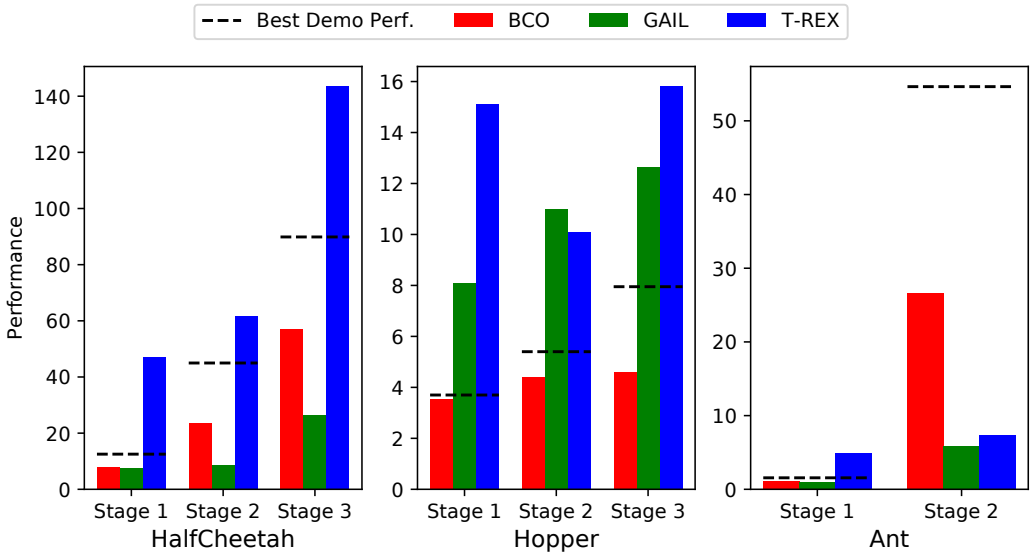
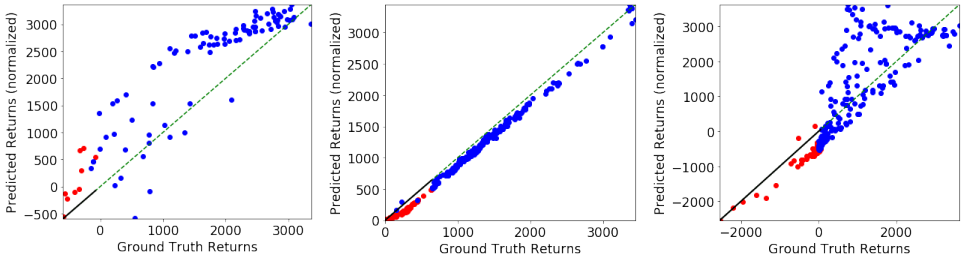


Figure 5.2: Imitation learning performance for three robotic locomotion tasks when given suboptimal demonstrations. Performance is measured as the total distance traveled, as measured by the final x-position of the robot's body. For each stage and task, the best performance given suboptimal demonstrations is shown for T-REX (ours), BCO (Torabi et al., 2018a), and GAIL (Ho and Ermon, 2016). The dashed line shows the performance of the best demonstration.

Reward Extrapolation

We next investigated the ability of T-REX to accurately extrapolate beyond the demonstrator. To do so, we compared ground-truth return and T-REX-inferred return across trajectories from a range of performance qualities, including trajectories much better than the best demonstration given to T-REX. The extrapolation of the reward function learned by T-REX is shown in Figure 5.3. The plots in Figure 5.3 give insight into the performance of T-REX. When T-REX learns a reward function that has a strong positive correlation with the ground-truth reward function, then it is able to surpass the performance of the suboptimal demonstrations. However, in Ant the correlation is not as strong, resulting in worse-than-demonstrator performance in Stage 2.



(a) HalfCheetah (b) Hopper (c) Ant

Figure 5.3: Extrapolation plots for T-REX on MuJoCo Stage 1 demonstrations. Red points correspond to demonstrations and blue points correspond to trajectories not given as demonstrations. The solid line represents the performance range of the demonstrator, and the dashed line represents extrapolation beyond the demonstrator's performance. The x-axis is the ground-truth return and the y-axis is the predicted return from our learned reward function. Predicted returns are normalized to have the same scale as the ground-truth returns.

5.2.4 Atari Experiments and Results

Demonstrations

We next evaluated T-REX on eight Atari games shown in Table 5.1. To obtain a variety of suboptimal demonstrations, we generated 12 full-episode trajectories using PPO policies checkpointed every 50 training updates for all games except for Seaquest and Enduro. For Seaquest, we used every 5th training update due to the ability of PPO to quickly find a good policy. For Enduro, we used every 50th training update starting from step 3,100 since PPO obtained 0 return until after 3,000 steps. We used the OpenAI Baselines implementation of PPO with the default hyperparameters.

Experimental Setup

We used an architecture for reward learning similar to the one proposed in (Ibarz et al., 2018), with four convolutional layers with sizes 7x7, 5x5, 3x3, and 3x3, with strides 3, 2, 1, and 1. Each convolutional layer used 16 filters and LeakyReLU non-linearities. We then used a fully connected layer with 64 hidden units and a single scalar output. We fed

in stacks of 4 frames with pixel values normalized between 0 and 1 and masked the game score and number of lives.

For all games except Enduro, we subsampled 6,000 trajectory pairs between 50 and 100 observations long. We optimized the reward functions using Adam with a learning rate of 5e-5 for 30,000 steps. Given two full trajectories τ_i and τ_j such that $\tau_i \prec \tau_j$, we first randomly sample a subtrajectory from τ_i. Let t_i be the starting timestep for this subtrajectory. We then sample an equal length subtrajectory from τ_j such that $t_i \leq t_j$, where t_j is the starting time step of the subtrajectory from τ_j. We found that this resulted in better performance than comparing randomly chosen subtrajectories, likely due to the fact that (1) it eliminates pairings that compare a later part of a worse trajectory with an earlier part of a better trajectory and (2) it encourages reward functions that are monotonically increasing as progress is made in the game. For Enduro, training on short partial trajectories was not sufficient to score any points and instead we used 2,000 pairs of down-sampled full trajectories (see appendix for details).

We optimized a policy by training a PPO agent on the learned reward function. To reduce reward scaling issues, we normalized predicted rewards by feeding the output of $\hat{r}_\theta(s)$ through a sigmoid function before passing it to PPO. We trained PPO on the learned reward function for 50 million frames to obtain our final policy. We also compare against Behavioral Cloning from Observation (BCO) (Torabi et al., 2018a) and the state-of-the-art Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016). Note that we give action labels to GAIL, but not to BCO or T-REX. We tuned the hyperparameters for GAIL to maximize performance when using expert demonstrations on Breakout and Pong. We gave the same demonstrations to both BCO and T-REX; however, we found that GAIL was very sensitive to poor demonstrations so we trained GAIL on 10 demonstrations using the policy checkpoint that generated the best demonstration given to T-REX.

Table 5.1: Comparison of T-REX with a state-of-the-art behavioral cloning algorithm (BCO) (Torabi et al., 2018a) and state-of-the-art IRL algorithm (GAIL) (Ho and Ermon, 2016). Performance is evaluated on the ground-truth reward. T-REX achieves better-than-demonstrator performance on 7 out of 8 games and surpasses the BCO and GAIL baselines on 7 out of 8 games. Results are the best average performance over three random seeds with 30 trials per seed.

	Ranked Demonstrations		LfD Algorithm Performance		
Game	Best	Average	T-REX	BCO	GAIL
Beam Rider	1,332	686.0	**3,335.7**	568	355.5
Breakout	32	14.5	**221.3**	13	0.28
Enduro	84	39.8	**586.8**	8	0.28
Hero	**13,235**	6,742.0	0	2,167	0
Pong	-6	-15.6	**-2.0**	-21	-21
Q*bert	800	627	**32,345.8**	150	0
Seaquest	600	373.3	**747.3**	0	0
Space Invaders	600	332.9	**1,032.5**	88	370.2

Learned Policy Performance

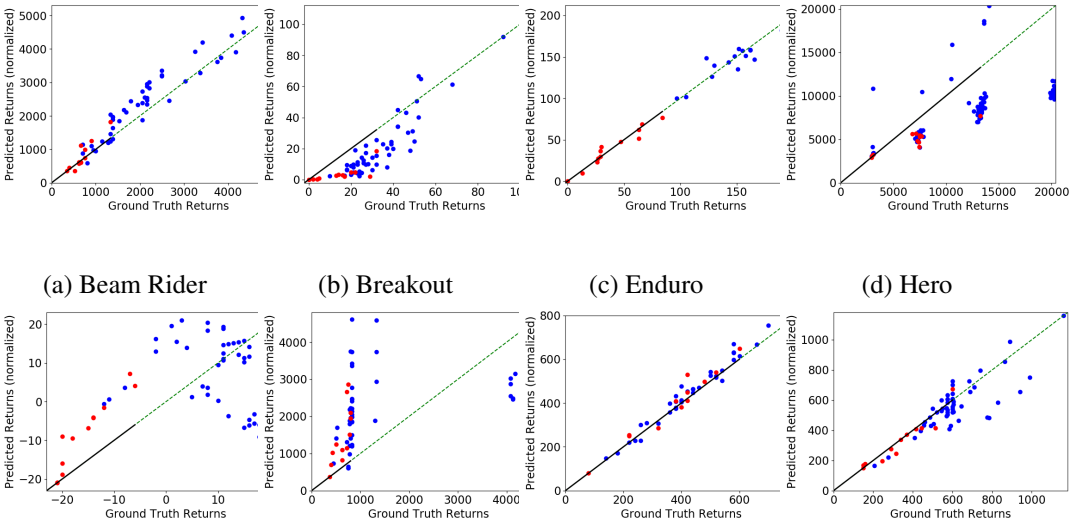
The average performance of T-REX under the ground-truth reward function and the best and average performance of the demonstrator are shown in Table 5.1. Table 5.1 shows that T-REX outperformed both BCO and GAIL in 7 out of 8 games. T-REX also outperformed the best demonstration in 7 out of 8 games. On four games (Beam Rider, Breakout, Enduro, and Q*bert) T-REX achieved score that is more than double the score of the best demonstration. In comparison, BCO performed worse than the average performance of the demonstrator in all games, and GAIL only performed better than the average demonstration on Space Invaders. Despite using better training data, GAIL was unable to learn good policies on any of the Atari tasks. These results are consistent with those of Tucker et al. (2018) that show that current GAN-based IRL methods do not perform well on Atari. In the appendix, we compare our results against prior work (Ibarz et al., 2018) that uses demonstrations plus active feedback during policy training to learn control policies for the Atari domain.

Reward Extrapolation

We also examined the extrapolation of the reward function learned using T-REX. Results are shown in Figure 5.4. We observed accurate extrapolation for Beam Rider, Breakout, Enduro, Seaquest, and Space Invaders—five games where T-REX significantly outperform the demonstrator. The learned rewards for Pong, Q*bert, and Hero show less correlation. On Pong, T-REX overfits to the suboptimal demonstrations and ends up preferring longer games which do not result in a significant win or loss. T-REX is unable to score any points on Hero, likely due to poor extrapolation and the higher complexity of the game. Surprisingly, the learned reward function for Q*bert shows poor extrapolation, yet T-REX is able to outperform the demonstrator. We analyzed the resulting policy for Q*bert and found that PPO learns a repeatable way to score points by inducing Coily to jump off the edge. This behavior was not seen in the demonstrations.

Reward Function Visualizations

To gain insights into the learned reward function, we plotted the maximum and minimum predicted observations from the trajectories used to create Figure 5.4 along with attention maps for the learned reward functions. For example, Figure C.2 shows the frame stack for Beam Rider with maximum and minimum predicted rewards. Figure 5.5 (a) shows the maximum predicted framestack where the agent destroys an enemy ship. Figure 5.5 (c) shows the framestack that had the smallest predicted reward. This observation shows the agent right as it is hit by an enemy bullet. Figure 5.6 shows the observations with maximum and minimum predicted scores. Even though none of the demonstrations were able to fully clear a level of aliens, the learned reward function is able to extrapolate and predict that clearing all of the aliens from the screen has highest reward. The lowest predicted score is not when the agent loses a life, but when it first starts the game and has not destroyed any of the alien ships. In Appendix C.7 we provide more results and the corresponding attention heatmaps for all of the games.



(a) Beam Rider (b) Breakout (c) Enduro (d) Hero

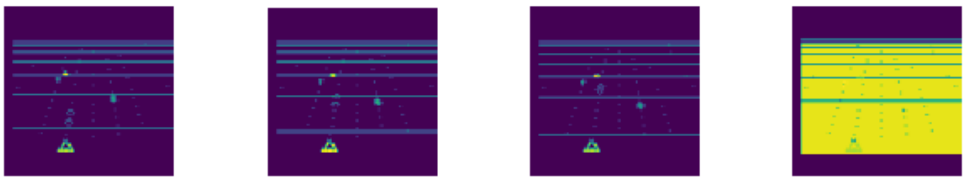
(e) Pong (f) Q*bert (g) Seaquest (h) Space Invaders

Figure 5.4: Extrapolation plots for Atari games. We compare ground truth returns over demonstrations to the predicted returns using T-REX (normalized to be in the same range as the ground truth returns). The black solid line represents the performance range of the demonstrator. The green dashed line represents extrapolation beyond the range of the demonstrator's performance.

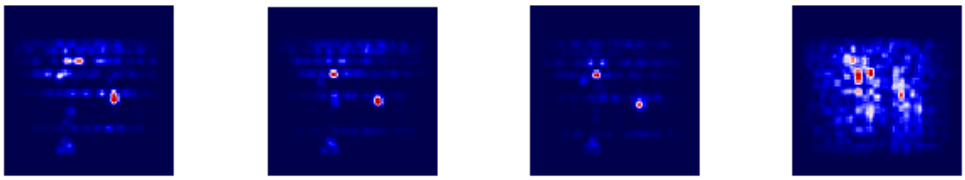
Human Demonstrations

The above results used synthetic demonstrations generated from an RL agent. We also tested T-REX when given ground-truth rankings over human demonstrations. We used novice human demonstrations from the Atari Grand Challenge Dataset (Kurin et al., 2017) for five Atari tasks. We used the ground truth returns in the Atari Grand Challenge data set to rank demonstrations. To generate demonstrations we removed duplicate demonstrations (human demonstrations that achieved the same score). We then sorted the remaining demonstrations based on ground truth return and selected 12 of these demonstrations to form our training set. We ran T-REX using the same hyperparameters as described above.

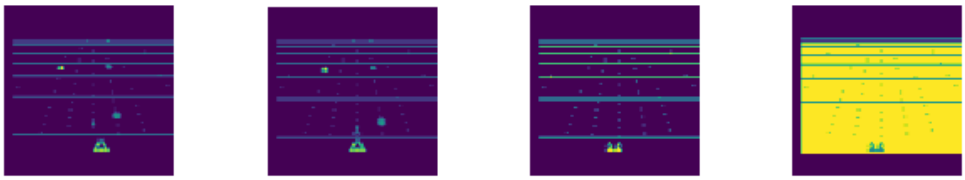
The resulting performance of T-REX is shown in Table 5.2. T-REX is able to outperform the best human demonstration on Q*bert, Space Invaders, and Video Pinball; however,



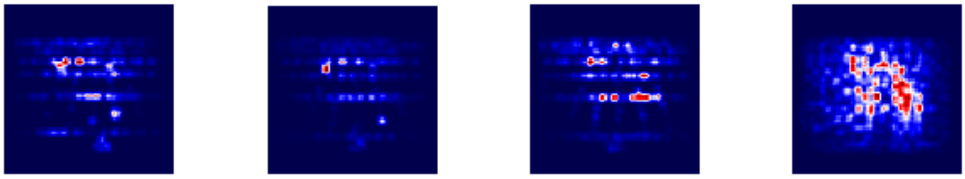
(a) Beam Rider observation with maximum predicted reward



(b) Beam Rider reward model attention on maximum predicted reward



(c) Beam Rider observation with minimum predicted reward

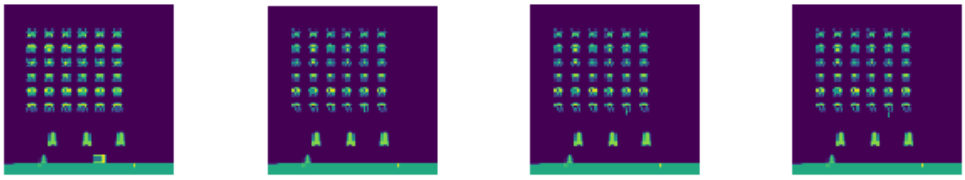


(d) Beam Rider reward model attention on minimum predicted reward

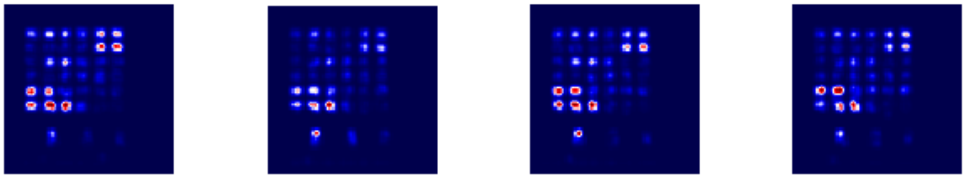
Figure 5.5: Maximum and minimum predicted observations and corresponding attention maps for Beam Rider. The observation with the maximum predicted reward shows successfully destroying an enemy ship, with the network paying attention to the oncoming enemy ships and the shot that was fired to destroy the enemy ship. The observation with minimum predicted reward shows an enemy shot that destroys the player's ship and causes the player to lose a life. The network attends most strongly to the enemy ships but also to the incoming shot.

(a) Space Invaders observation with maximum predicted reward

(b) Space Invaders reward model attention on maximum predicted reward



(c) Space Invaders observation with minimum predicted reward



(d) Space Invaders reward model attention on minimum predicted reward

Figure 5.6: Maximum and minimum predicted observations and corresponding attention maps for Space Invaders. The observation with maximum predicted reward shows an observation where all the aliens have been successfully destroyed and the protective barriers are still intact. Note that the agent never observed a demonstration that successfully destroyed all the aliens. The attention map shows that the learned reward function is focused on the barriers, but does not attend to the location of the controlled ship. The observation with minimum predicted reward shows the very start of a game with all aliens still alive. The network attends to the aliens and barriers, with higher weight on the aliens and barrier closest to the space ship.

Table 5.2: T-REX performance with real novice human demonstrations collected from the Atari Grand Challenge Dataset Kurin et al. (2017). Results are the best average performance over three random seeds with 30 trials per seed.

Game	Novice Human Best	Novice Human Average	T-REX
Montezuma's Revenge	**2,600**	1,275.0	0.0
Ms Pacman	**1,360**	818.3	550.7
Q*bert	875	439.6	**6,869.2**
Space Invaders	470	290.0	**1,092.0**
Video Pinball	4,210	2,864.3	**20,000.2**

it is not able to learn a good control policy for Montezuma's Revenge or Ms Pacman. These games require maze navigation and balancing different objectives, such as collecting objects and avoiding enemies. This matches our results in the main text that show that T-REX is unable to learn a policy for playing Hero, a similar maze navigation task with multiple objectives such as blowing up walls, rescuing people, and destroying enemies. Extending T-REX to work in these types of settings is an interesting area of future work.

5.2.5 Robustness to Noisy Rankings

All experiments described thus far have had access to ground-truth rankings. To explore the effects of noisy rankings we first examined the stage 1 Hopper task. We synthetically generated ranking noise by starting with a list of trajectories sorted by ground-truth returns and randomly swapping adjacent trajectories. By varying the number of swaps, we were able to generate different noise levels. Given n trajectories in a ranked list provides $\binom{n}{2}$ pairwise preferences over trajectories. The noise level is measured as a total order correctness: the fraction of trajectory pairs whose pairwise ranking after random swapping matches the original ground-truth pairwise preferences. The results of this experiment, averaged over 9 runs per noise level, are shown in Figure 5.7. We found that T-REX is relatively robust to noise of up to around 15% pairwise errors.

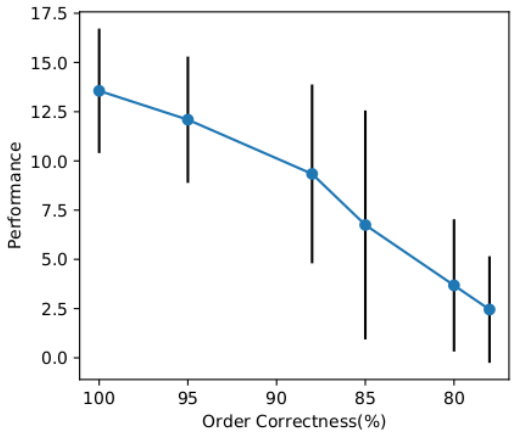


Figure 5.7: The performance of T-REX for different amounts of pairwise ranking noise in the Hopper domain. T-REX shows graceful degradation as ranking noise increases. The reward function is trained on stage-1 Hopper demonstrations. The graph shows the mean across nine trials and 95% confidence interval.

To examine the effect of noisy human rankings, we used the synthetic PPO demonstrations that were used in the previous Atari experiments and used Amazon Mechanical Turk to collect human rankings. We presented videos of the demonstrations in pairs along with a brief text description of the goal of the game and asked workers to select which demonstration had better performance, with an option for selecting "Not Sure". We collected six labels per demonstration pair and used the most-common label as the label for training the reward function. We removed from the training data any pairings where there was a tie for the most-common label or where "Not Sure" was the most common label. We found that despite this preprocessing step, human labels added a significant amount of noise and resulted in pair-wise rankings with accuracy between 63% and 88% when compared to ground-truth labels. However, despite significant ranking noise, T-REX outperformed the demonstrator on 5 of the 8 Atari games as shown in Table 5.3.

The resulting accuracy and number of labels that had a majority preference are shown in Table 5.3. We ran T-REX using the same hyperparameters described in the main text. We ran PPO with 3 different seeds and report the performance of the best final policy

averaged over 30 trials. We found that surprisingly, T-REX is able to optimize good policies for many of the games, despite noisy labels. However, we did find cases such as Enduro, where the labels were too noisy to allow successful policy learning.

Table 5.3: Evaluation of T-REX on human rankings collected using Amazon Mechanical Turk. Results are the best average performance over three random seeds with 30 trials per seed.

	Human-Ranked Demonstrations				
Game	Best	Average	Ranking Accuracy	Num. Labels	T-REX avg. perf.
Beam Rider	1,332	686.0	63.0%	54	**3,457.2**
Breakout	32	14.5	88.1%	59	**253.2**
Enduro	**84**	39.8	58.6%	58	0.03
Hero	**13,235**	6742	77.6%	58	2.5
Pong	**-6**	-15.6	79.6%	54	-13.0
Q*bert	800	627	75.9%	58	**66,082**
Seaquest	600	373.3	80.4%	56	**655.3**
Space Invaders	600	332.9	84.7%	59	**1,005.3**

5.2.6 Discussion

This chapter focuses on efficient, better-than-demonstrator imitation learning. In the preceeding sections, we introduced T-REX, a reward learning technique for high-dimensional tasks that can learn to extrapolate intent from suboptimal ranked demonstrations. T-REX learns a reward function without requiring an MDP solver or any data collection during reward inference. To the best of our knowledge, this is the first IRL algorithm that is able to significantly outperform the demonstrator and that scales to high-dimensional Atari games. When combined with deep reinforcement learning, we showed that this approach achieves better-than-demonstrator performance as well as outperforming state-of-the-art behavioral cloning and IRL methods. We also demonstrated that T-REX is robust to modest amounts of ranking noise. In the next section we explore how to apply T-REX to the more general imitation learning settings where explicit preference labels are unavailable.

5.3 Ranking-Based Reward Extrapolation Without Rankings

In this section we explore two different methods for efficient, better-than-demonstrator imitation learning without explicit rankings. We first explore the case where an agent is learning from a demonstrator that is also learning. Many user studies involving imitation learning give demonstrators a practice phase where a user gets familiar with the task and is able to learn how to give good demonstrations. In this section we give evidence that suboptimal demonstrations obtained during "practice rounds" with human demonstrators should not be simply thrown away as they contain useful information about failure modes. We show that an imitation learning algorithm can learn to extrapolate the performance of a learning agent simply by watching an agent's behavior as it learns how to perform a task better over time.

Next, we introduce Disturbance-based Reward Extrapolation (D-REX), an extension of T-REX to the more general imitation learning setting where the learning agent is given a bag of demonstrations with no preference labels or timestamps. In this case we show that an imitation learning agent can first perform behavioral cloning on the demonstrations and then inject noise injection into the cloned policy to self-generate a wide variety of automatically-ranked demonstrations which can be used by T-REX to learn a better-than-demonstrator policy in the absence of explicit preferences. We empirically validate our approach on simulated robot and Atari imitation learning benchmarks and show that D-REX outperforms standard imitation learning approaches and can significantly surpass the performance of the demonstrator. D-REX is the first imitation learning approach to achieve significant extrapolation beyond the demonstrator's performance without additional side-information or supervision, such as rewards or human preferences. By generating rankings automatically, we show that preference-based inverse reinforcement learning can be applied in traditional imitation learning settings where only unlabeled demonstrations are available.

5.3.1 Learning from a Learner

In this section we focus on learning a reward function simply by watching a learning agent.

Motivation

Why should we be able to learn a reward function from simply watching a learner? Here we give some intuitive examples as to why it is theoretically possible, under strong assumptions, to learn from watching a learner, should be possible in theory. Later we discuss a practical approach for learning from a learner.

As a simple example of a simple case where learning from a learner is possible, consider the learning update of the Q-learning algorithm Sutton and Barto (1998):

$$\hat{Q}_{t+1}(s,a) = \hat{Q}_t(s,a) + \alpha[R(s) + \gamma \max_{a'} \hat{Q}_t(s',a') - \hat{Q}_t(s,a)]. \tag{5.11}$$

By rearranging terms we see that the reward can be recovered if we have access to the Q-values of the learner.

$$R(s) = \frac{\hat{Q}_{t+1}(s,a) - \hat{Q}_t(s,a)}{\alpha} + \hat{Q}_t(s,a) - \gamma \max_{a'} \hat{Q}_t(s',a') \tag{5.12}$$

Similarly, consider a parameterized policy, π_θ, being learned through policy gradient RL Sutton and Barto (1998). Given trajectories $\tau \sim \pi_\theta$, a basic update using the policy gradient theorem (Sutton et al., 2000) can be written as:

$$\theta_{t+1} = \theta_t + \alpha \mathbb{E}_{\pi_\theta}\left[r(\tau) \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)\right]. \tag{5.13}$$

Rearranging terms and assuming a stochastic gradient descent update using a single policy trajectory $\tau \sim \pi_\theta$ we can recover the return along the trajectory

$$r(\tau) = \frac{\theta_{t+1} - \theta_t}{\alpha \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)} \tag{5.14}$$

Thus, if we have access to the internals of the learning agent and its changes in policy, information about the reward can be learned long before the optimal policy is learned.

This fact is also clear from the success of model-based RL (Deisenroth and Rasmussen, 2011; Chua et al., 2018), where the goal is to build a model of the transition dynamics and reward function in order to reduce the real-world sample complexity of solving an MDP; reward functions are often much easier to learn than optimal policies. Further, by learning a parameterized reward function from environmental features, we can learn a reward function that generalizes to areas of state-space that are unvisited by demonstrations.

While having direct access to the demonstrator's internal state is implausible—and would likely mean we have access to the true reward signal as well—we do have access to the trajectories taken by the learner which are informed by its internal state, i.e., reward signals, Q-values, policy-gradient, etc. The goal of this section is to develop a learning algorithm that can leverage the changes in trajectories taken by a learner in order to estimate a hidden reward signal. In doing so, we can then learn a policy that can surpass the performance of the demonstrator. The ideas in this section were originally presented in (Brown et al., 2019b). Concurrent with our work, Jacq et al. (2019) also proposed to study the problem of learning from a learner. While the work of Jacq et al. (2019) is motivated by the same principle—a learning agent's performance improves over time—they take a very different approach by first learning a series of policies via imitation learning and then using those policies to estiamte the reward function using entropy regularized policy iteration.

Learning from a Learner via T-REX

In the remainder of this section we investigate whether T-REX can learn good control policies simply by observing a novice demonstrator that noisily improves at a task over time. We assume that the demonstrator is a learning agent that receives a hidden reward signal and uses that signal to learn how to perform a task. Note that the agent labeled as the demonstrator may be a human or artificial agent and does not need to know that they are being observed or that their behavior will be used for imitation learning. In this case we do not have explicit preference rankings. However, under the assumption that the trajectories

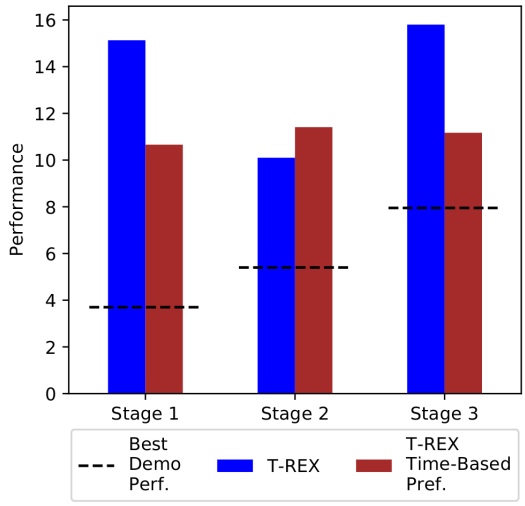


Figure 5.8: T-REX results with time-based rankings in the Hopper domain.

are generated by a learning agent, we make the assumption that, on average, the agent's performance at the task is improving. Thus, we can apply weak labels based on timestamps to prefer later trajectories over earlier trajectories. Given these weakly labeled demonstrations, we can learn a reward function and control policy via standard T-REX as described in the previous section.

To test this approach, we took the same demonstrations used for the MuJoCo tasks, and rather than sorting them based on ground-truth rankings, we used the order in which they were generated by PPO to produce a ranked list of trajectories, ordered by timestamp from earliest to latest. This provides ranked demonstrations without any need for demonstrator labels, and enables us to test whether simply observing an agent learn over time allows us to extrapolate intention by assuming that later trajectories are preferable to trajectories produced earlier in learning. The results for Hopper are shown in Figure 5.8. Table 5.4 shows the full results for the MuJoCo experiments comparing the time-ordered learning from a learner results with the explicit preference ranking results from the previous section. The T-REX (time-ordered) row shows the resulting performance of T-REX when demonstrations come from observing a learning agent and are ranked based on timestamps

Table 5.4: The results on three robotic locomotion tasks when given suboptimal demonstrations. Performance is measured as the total distance traveled, as measured by the final x-position of the robot's body. For each stage and task, the best performance given suboptimal demonstrations is shown on the top row, and the best achievable performance (i.e. performance achieved by a PPO agent) under the ground-truth reward is shown on the bottom row. The mean and standard deviation are based on 25 trials (obtained by running PPO five times and for each run of PPO performing five policy rollouts). The first row of T-REX results show the performance when demonstrations are ranked using the ground-truth returns. The second row of T-REX shows results for learning from observing a learning agent (time-ordered). The demonstrations are ranked based on the timestamp when they were produced by the PPO algorithm learning to perform the task.

	HalfCheetah			Hopper			Ant	
	Stage 1	Stage 2	Stage 3	Stage 1	Stage 2	Stage 3	Stage 1	Stage 2
Best Demo	12.52	44.98	89.87	3.70	5.40	7.95	1.56	**54.64**
Performance	(1.04)	(0.60)	(8.15)	(0.01)	(0.12)	(1.64)	(1.28)	(22.09)
T-REX	46.90	**61.56**	143.40	**15.13**	10.10	**15.80**	4.93	7.34
(ours)	(1.89)	(10.96)	(3.84)	(3.21)	(1.68)	(0.37)	(2.86)	(2.50)
T-REX	**51.39**	54.90	**154.67**	10.66	**11.41**	11.17	**5.55**	1.28
(time-ordered)	(4.52)	(2.29)	(57.43)	(3.76)	(0.56)	(0.60)	(5.86)	(0.28)
BCO	7.71	23.59	57.13	3.52	4.41	4.58	1.06	26.56
	(8.35)	(8.33)	(19.14)	(0.14)	(1.45)	(1.07)	(1.79)	(12.96)
GAIL	7.39	8.42	26.28	8.09	10.99	12.63	0.95	5.84
	(4.12)	(3.43)	(12.73)	(3.25)	(2.35)	(3.66)	(2.06)	(4.08)
Best w/		199.11			15.94		182.23	
GT Reward		(9.08)			(1.47)		(8.98)	

rather than using explicit preference rankings. We found that T-REX is able to infer a meaningful reward function even when noisy, time-based rankings are provided. All the trained policies produced comparable results on most stages to the ground-truth rankings, and those policies outperform BCO and GAIL on all tasks and stages except for Ant Stage 2.

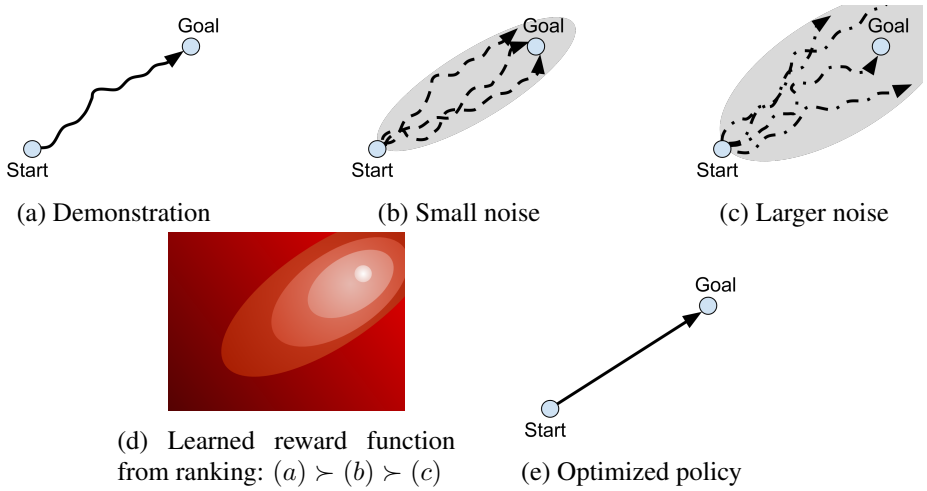
5.3.2 Disturbance-Based Reward Extrapolation

Previously in this chapter, we presented theoretical results for when better-than-demonstrator performance is possible and demonstrated theoretically that rankings (or alternatively, pairwise preferences) over demonstrations can enable better-than-demonstrator performance by reducing error and ambiguity in the learned reward function. We then proposed Trajectory-

ranked Reward Extrapolation (T-REX) as an efficient algorithm that can achieve better-than-demonstrator performance via preference rankings over demonstrations. Our previous results demonstrate that T-REX is applicable to the case where explicit preference rankings are unavailable, but where the observed trajectories come from a learning agent and can be weakly labeled based on timestamps.

In this section, we address the problem of leveraging the benefits of reward learning via ranked demonstrations in a way that does not require human rankings. T-REX learns a reward function that allows better-than-demonstrator performance without requiring human supervision during policy learning. However, requiring a demonstrator to rank demonstrations can be tedious and error prone, and precludes learning from prerecorded, unranked demonstrations, or learning from demonstrations of similar quality that are difficult to rank. In this section, we investigate whether it is possible to generate a set of ranked demonstrations, in order to surpass the performance of a demonstrator, without requiring supervised preference labels or reward information.

We propose Disturbance-based Reward Extrapolation (D-REX), a ranking-based reward learning algorithm that does not require ranked demonstrations. Our approach injects noise into a policy learned through behavioral cloning to automatically generate ranked policies of varying performance. D-REX makes the weak assumption that the demonstrations are better than a purely random policy, and that adding increasing levels of noise into a cloned policy will result in increasingly worse performance, converging to a random policy in the limit. Our approach is summarized in Figure 5.9. The intuition behind this approach is that generating ranked trajectories via noise injection reveals relative weightings between reward features: features that are more prevalent in noisier trajectories are likely inversely related to the reward, whereas features that are more common in noise-free trajectories are likely features which are positively correlated with the true reward. Furthermore, adding noise provides a form of feature selection since, if a feature is equally common across all levels of noise, then it likely has no impact on the true reward function and can be ignored.



(a) Demonstration (b) Small noise (c) Larger noise

(d) Learned reward function from ranking: $(a) \succ (b) \succ (c)$ (e) Optimized policy

Figure 5.9: **D-REX high-level approach:** given a suboptimal demonstration (a), we run behavioral cloning to approximate the demonstrator's policy. By progressively adding more noise to this cloned policy ((b) and (c)), we are able to automatically synthesize a preference ranking: $(a) \succ (b) \succ (c)$. Using this ranking, we learn a reward function (d) which is then optimized using reinforcement learning to obtain a policy (e) that performs better than the demonstrator.

By automatically generating rankings, preference-based imitation learning methods (Sadigh et al., 2017; Ibarz et al., 2018; Palan et al., 2019; Brown et al., 2019b) can be applied in standard imitation learning domains where rankings are unavailable. We demonstrate this by combining automatic rankings via noise-injections with T-REX (Section 5.2). We empirically validate our approach on simulated robotics and Atari benchmarks and find that D-REX results in policies that can both significantly outperform the demonstrator as well as significantly outperform standard imitation learning. To the best of our knowledge, D-REX is the first imitation learning approach to achieve significant performance improvements over the demonstrations without requiring extra supervision or additional side-information, such as ground-truth rewards or human preferences.

Algorithm 3 D-REX: Disturbance-based Reward Extrapolation

Require: Demonstrations $\mathcal{D}$, noise schedule $\mathcal{E}$, number of rollouts K

1: Run behavioral cloning on demonstrations $\mathcal{D}$ to obtain policy π_{BC}
2: **for** $\epsilon_i \in \mathcal{E}$ **do**
3: Generate a set of K trajectories from a noise injected policy $\pi_{\text{BC}}(\cdot|\epsilon_i)$
4: Generate automatic preference labels $\tau_i \prec \tau_j$ if $\tau_i \sim \pi_{\text{BC}}(\cdot|\epsilon_i)$, $\tau_j \sim \pi_{\text{BC}}(\cdot|\epsilon_j)$, and $\epsilon_i > \epsilon_j$
5: Run T-REX (Brown et al., 2019b) on automatically ranked trajectories to obtain $\hat{R}$
6: Optimize policy π using reinforcement learning with reward function $\hat{R}$
7: **return** π

5.3.3 Algorithm

We now describe Disturbance-based Reward Extrapolation (D-REX), our proposed approach for automatically generating ranked demonstrations. Our approach is summarized in Algorithm 3. Videos of the learned policies as well as the code to reproduce our results are available online.[9]

We first take a set of unranked demonstrations and use behavioral cloning to learn a policy π_{BC}. Behavioral cloning (Bain and Sommut, 1999) treats each state action pair $(s, a) \in \mathcal{D}$ as a training example and seeks a policy π_{BC} that maps from states to actions. We model π_{BC} using a neural network with parameters θ_{BC} and find these parameters using maximum-likelihood estimation such that $\theta_{\text{BC}} = \arg\max_\theta \prod_{(s,a)\in\mathcal{D}} \pi_{\text{BC}}(a|s)$. By virtue of the optimization procedure, π_{BC} will usually only perform as well as the average performance of the demonstrator—at best it may perform slightly better than the demonstrator if the demonstrator makes mistakes approximately uniformly at random.

Our main insight is that if π_{BC} is significantly better than the performance of a completely random policy, then we can inject noise into π_{BC} and interpolate between the performance of π_{BC} and the performance of a uniformly random policy. In Appendix B.5, we prove that given a noise schedule $\mathcal{E} = (\epsilon_1, \epsilon_2, \ldots, \epsilon_d)$ consisting of a sequence of noise levels such that $\epsilon_1 > \epsilon_2 > \ldots > \epsilon_d$, then with high-probability, $J(\pi_{\text{BC}}(\cdot|\epsilon_1)) <$

[9] The project website and code can be found at https://dsbrown1331.github.io/CoRL2019-DREX/

$J(\pi_{\text{BC}}(\cdot|\epsilon_2)) < \cdots < J(\pi_{\text{BC}}(\cdot|\epsilon_d))$. Given noise level $\epsilon \in \mathcal{E}$, we inject noise via an ϵ-greedy policy such that with probability 1-ϵ, the action is chosen according to π_{BC}, and with probability ϵ, the action is chosen uniformly at random within the action range.

For every ϵ, we generate K policy rollouts and thus obtain $K \times d$ ranked demonstrations, where each trajectory is ranked based on the noise level that generated it, with trajectories considered of equal preference if generated from the same noise level. Thus, by generating rollouts from $\pi_{\text{BC}}(\cdot|\epsilon)$ with varying levels of noise, we can obtain an arbitrarily large number of ranked demonstrations:

$$D_{\text{ranked}} = \{\tau_i \prec \tau_j : \tau_i \sim \pi_{\text{BC}}(\cdot|\epsilon_i), \tau_j \sim \pi_{\text{BC}}(\cdot|\epsilon_j), \epsilon_i > \epsilon_j\}. \tag{5.15}$$

Given these auto-generated ranked demonstrations, we then use T-REX to learn a reward function $\hat{R}$. Similar to T-REX, we use the following pairwise ranking loss function (Cao et al., 2007) based on the Bradley-Terry-Luce-Shephard choice rule (Luce, 2012):

$$\mathcal{L}(\theta) \approx -\frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log \frac{\exp \sum_{s\in\tau_j} \hat{R}_\theta(s)}{\exp \sum_{s\in\tau_i} \hat{R}_\theta(s) + \exp \sum_{s\in\tau_j} \hat{R}_\theta(s)}, \tag{5.16}$$

where $\mathcal{P} = \{(i,j) : \tau_i \prec \tau_j\}$.[10] After learning a reward function, we can optimize a policy $\hat{\pi}$ using any reinforcement learning algorithm. In the experiments below we optimize policies using the PPO algorithm Schulman et al. (2017) (see Appendix D.1 for details).

5.3.4 Experimental Results

Automatically generating rankings via noise

To test whether injecting noise can create high-quality, automatic rankings, we used simulated suboptimal demonstrations from a partially trained reinforcement learning agent. To

[10] Note that this can be implemented efficiently and robustly in most automatic differentiation software packages by simply using a binary cross entropy loss function where the logits are the predicted cumulative returns.

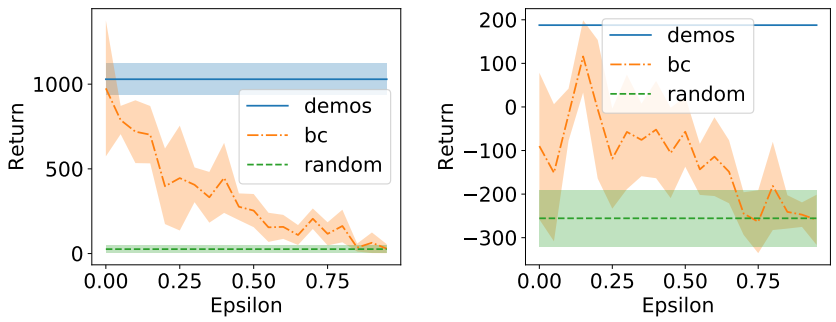
do so, we used the PPO implementation from OpenAI Baselines Dhariwal et al. (2017) to partially train a policy on the ground-truth reward function. We then ran behavioral cloning on these demonstrations and plotted the degradation in policy performance for increasing values of ϵ.

We evaluated noise degradation on the Hopper and Half-Cheetah domains in MuJoCo and on the seven Atari games listed in Table 5.5. To perform behavioral cloning, we used one suboptimal demonstration trajectory of length 1,000 for the MuJoCo tasks and 10 suboptimal demonstrations for the Atari games. We then varied ϵ and generated rollouts for different noise levels. We plotted the average return along with one standard deviation error bars in Figure 5.10 (see Appendix D.1.4 for details). We found that behavioral cloning with small noise tends to have performance similar to that of the average performance of the demonstrator. As noise is added, the performance degrades until it reaches the level of a uniformly random policy ($\epsilon = 1$). These plots validate our assumption that, in expectation, adding increasing amounts of noise will cause near-monotonic performance degradation.

Reward extrapolation

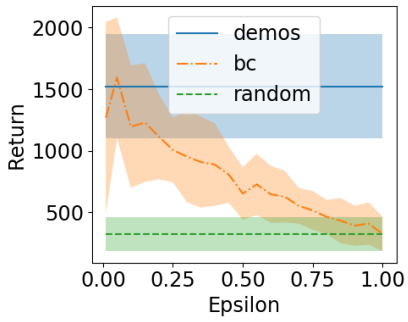
We next tested whether D-REX allows for accurate reward extrapolation. We used noise injection, as described in the previous section, to generate 100 synthetically-ranked demonstrations. For MuJoCo, we used the noise schedule consisting of 20 different noise levels, evenly spaced over the interval $[0, 1)$ and generated $K = 5$ rollouts per noise level. For Atari, we used the noise schedule $\mathcal{E} = (1.0, 0.75, 0.5, 0.25, 0.02)$ with $K = 20$ rollouts per noise level. By automatically generating ranked demonstrations, D-REX is able to leverage a small number of unranked demonstrations to generate a large dataset of ranked demonstrations for reward function approximation. We used the T-REX algorithm (Brown et al., 2019b) to learn a reward function from these synthetically ranked demonstrations.

To investigate how well D-REX learns the true reward function, we evaluated the learned reward function $\hat{R}_\theta$ on the original demonstrations and the synthetic demonstrations

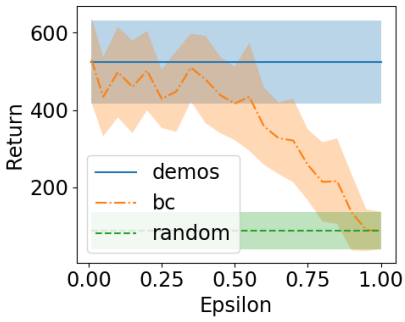


(a) Hopper

(b) HalfCheetah



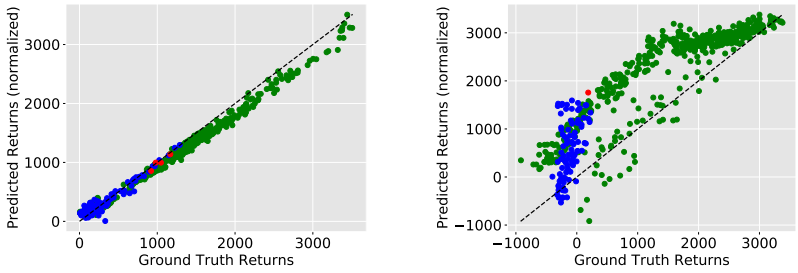
(c) Beam Rider



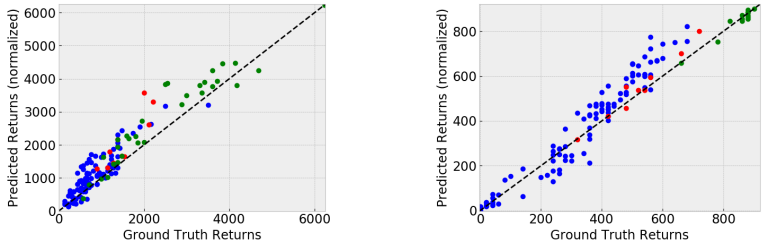
(d) Seaquest

Figure 5.10: Examples of the degradation in performance of an imitation policy learned via behavioral cloning as more noise is injected into the policy. Behavioral cloning is done on a 1,000-length trajectory (MuJoCo tasks) or 10 demonstrations (Atari games). Plots show mean and standard deviations over 5 rollouts (MuJoCo tasks) or 20 rollouts (Atari games).

obtained via noise injection. We then compared the ground-truth returns with the predicted returns under $\hat{R}_\theta$. We also tested reward extrapolation on a held-out set of trajectories obtained from PPO policies that were trained longer on the ground-truth reward than the policy used to generate the demonstrations for D-REX. These additional trajectories allow us to measure how well the learned reward function can extrapolate beyond the performance of the original demonstrations. The results for four of the tasks are shown in Figure 5.11. The remaining plots are included in Appendix D.3. The plots show relatively strong correlation between ground truth returns and predicted returns across most tasks, despite having no a priori access to information about true returns, nor rankings. We also generated re-



(a) Hopper (b) HalfCheetah



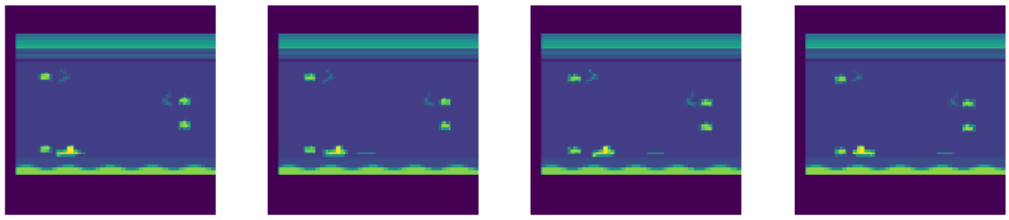
(c) Beam Rider (d) Seaquest

Figure 5.11: Extrapolation plots for a selection of MuJoCo and Atari tasks (see the appendix for more plots). Blue dots represent synthetic demonstrations generated via behavioral cloning with different amounts of noise injection. Red dots represent actual demonstrations, and green dots represent additional trajectories not seen during training. We compare ground truth returns over demonstrations to the predicted returns from D-REX (normalized to be in the same range as the ground truth returns).

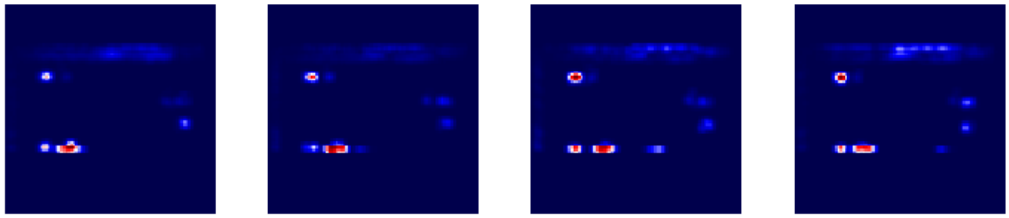
ward sensitivity heat maps (Greydanus et al., 2018) for the learned reward functions. These visualizations provide evidence that D-REX learns semantically meaningful features that are highly correlated with the ground truth reward. For example, Figure 5.12 shows that for Seaquest, the reward function learns a shaped reward that gives a large penalty for an imminent collision with an enemy (see Appendix D.3 for plots for all games).

Extrapolating beyond the demonstrator's performance

Next, we tested whether the reward functions learned using D-REX can be used in conjunction with deep reinforcement learning to achieve better-than-demonstrator performance. We ran PPO on the learned reward function $\hat{R}_\theta$ for 1 million timesteps (MuJoCo tasks) and 50



(a) Seaquest observation with minimum predicted reward using D-REX.



(b) Seaquest reward model attention on minimum predicted reward using D-REX.

Figure 5.12: D-REX minimum predicted observation and corresponding attention heatmap for Seaquest across a held-out set of 15 demonstrations. The observation with minimum predicted reward shows the submarine one frame before it is hit and destroyed by an enemy shark. This is an example of how the network has learned a shaped reward that helps it play the game better than the demonstrator. The network has learned to give most attention to nearby enemies and to the controlled submarine.

million frames (Atari games). We ran three replicates of PPO with different seeds and report the best performance on the ground-truth reward function, averaged over 20 trajectory rollouts. Table 5.5 compares the performance of the demonstrator with the performance of D-REX, behavioral cloning (BC), and Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016), a state-of-the-art imitation learning algorithm.

The results in Table 5.5 demonstrate that policies optimized using D-REX outperform the best demonstration in all tasks except for Pong. Furthermore, D-REX is also able to outperform BC and GAIL across all tasks except for Hopper and Pong. On the simulated MuJoCo robotics tasks, D-REX results in a 77% (Hopper) and 418% (HalfCheetah) performance increase when compared with the best demonstration. On Q*Bert, D-REX exploits a known loophole in the game which allows nearly infinite points. Excluding Q*Bert, D-REX results in an average performance increase of 39% across the Atari tasks, when compared

Table 5.5: Comparison of the performance of D-REX with behavioral cloning (BC), GAIL (Ho and Ermon, 2016), and the demonstrator's performance. Results are the best average ground-truth returns over three random seeds with 20 trials per seed. Bold denotes performance that is better than the best demonstration.

	Demonstrations		D-REX		BC		GAIL	
Task	Avg.	Best	Average	Stdev.	Avg.	Stdev.	Avg.	Stdev.
Hopper	1029	1167	**2072**	(1574)	943	(208)	**2700**	(692)
HalfCheetah	187	187	**972**	(96)	-115	(179)	85	(86)
Beam Rider	1,524	2,216	**7,220**	(2221)	1,268	(776)	1778	(787)
Breakout	34	59	**94**	(16)	29	(10)	0	(0)
Enduro	85	134	**247**	(88)	83	(27)	62	(24)
Pong	3	**14**	-9	(9)	8	(9)	-3.4	(3)
Q*bert	770	850	**22543**	(7434)	1,013	(721)	737	(311)
Seaquest	524	720	**801**	(4)	530	(109)	554	(108)
Space Invaders	538.5	930.0	**1,122.5**	(501.2)	426.5	(187.1)	364.8	(139.7)

with the best demonstration.

Worst-Case Performance Analysis

To test the robustness of the policy learned via D-REX, we also considered the worst-case performance, something that is important for safe inverse reinforcement learning (see Chapter 4). We investigated the worst-case performance of the demonstrator as it compares to the worst-case performance of the policy learned using D-REX. The results are shown in Table 5.6. Our results show that D-REX is able to learn safer policies than the demonstrator on 6 out of 7 games via intent extrapolation. The results show that on all games, except for Pong, D-REX is able to find a policy with both higher expected utility (see Table 1 in the main text) as well as higher worst-case utility (Table 5.6) when compared to the worst case performance of the demonstrator. D-REX also has a better worst-case performance than BC and GAIL across all games except for Pong.

Table 5.6: Comparison of the average and worst-case performance of D-REX with respect to the demonstrator. Results are the worst-case performance corresponding to the results shown in Table 1 in the main text. Bold denotes worst-case performance that is better than the worst-case demonstration.

	Worst-Case Performance			
Game	Demonstrator	D-REX	BC	GAIL
Beam Rider	900	**2916**	528	352
Breakout	17	**62**	13	0
Enduro	37	**152**	35	13
Pong	-5	-21	**-2**	-14
Q*bert	575	**7675**	650	350
Seaquest	320	**800**	280	260
Space Invaders	190	**575**	120	235

Live-Long Baseline

We noticed that for many Atari games, the goal is to stay alive as long as possible. To ensure that D-REX is learning more than a simple bonus for staying alive, we also compared D-REX with a PPO agent trained with a +1 reward for every timestep. Because we normalize the D-REX learned reward using a sigmoid, one concern is that the non-negativity of the sigmoid is the only thing that is needed to perform well on the Atari domain since games typically involve trying to stay alive as long as possible. We tested this by creating a live-long baseline that always rewards the agent with a +1 reward for every timestep. Table 5.7 shows that while a +1 reward is sufficient to achieve a moderate score on some games, it is insufficient to learn to play the games Enduro and Seaquest, both of which D-REX is able to learn to play. The reason that a +1 reward does not work on Enduro and Seaquest is that, in these games, it is possible to do nothing and cause an arbitrarily long episode. Thus, simply rewarding long episodes is not sufficient to learn to actually play. While a +1 reward was sufficient to achieve moderate to good scores on the other games, live-long reward is only able to surpass the performance of D-REX on Pong, which gives evidence that even if games where longer trajectories are highly correlated with the ground-truth

Table 5.7: Comparison of D-REX with other imitation learning approaches. BC is behavioral cloning. Live-long assigns every observation a +1 reward and is run using an experimental setup identical to D-REX.

	Demonstrator		Imitation method		
Game	Avg.	Best.	D-REX	Live-Long	BC
Beam Rider	1524	2216	**7220**	5583.5	1268.6
Breakout	34.5	59	**94.7**	68.85	29.75
Enduro	85.5	134	**247.9**	0	83.4
Pong	3.7	**14**	-9.5	-5.3	8.6
Q*bert	770	850	**22543.8**	17073.75	1013.75
Seaquest	524	720	**801**	1	530
Space Invaders	538.5	930	**1122.5**	624	426.5

return, D-REX is not simply rewarding longer episodes, but also rewarding trajectories that follow the demonstrators intention. This is also backed up by our reward attention heat maps in Appendix 8.5b, which demonstrate that D-REX is paying attention to details in the observations which are correlated with the ground-truth reward.

5.4 Summary

This chapter has focused on efficient reward inference algorithms that do not require an MDP solver in the inner-loop and can learn from suboptimal demonstrations. Imitation learning approaches are typically unable to outperform the demonstrator. This is because most approaches either directly mimic the demonstrator or find a reward function that makes the demonstrator appear near optimal. While algorithms that can exceed the performance of a demonstrator exist, they either rely on a significant number of active queries from a human (Christiano et al., 2017; Sadigh et al., 2017; Palan et al., 2019), a hand-crafted reward function (Hester et al., 2018), or pre-ranked demonstrations (Brown et al., 2019b). Furthermore, prior research has lacked theory about when better-than-demonstrator performance is possible. In this chapter we first addressed this lack of theory by presenting a

sufficient condition for extrapolating beyond the performance of a demonstrator. We also provided theoretical results demonstrating how preferences and rankings allow for better reward function learning by reducing the learner's uncertainty over the true reward function.

In this chapter, we introduced T-REX, a reward learning technique for complex, high-dimensional tasks that can learn to extrapolate intent from suboptimal ranked demonstrations. To the best of our knowledge, this is the first IRL algorithm that is able to significantly outperform the demonstrator without additional external knowledge (e.g. signs of feature contributions to reward) and that scales to high-dimensional Atari games. T-REX is able learn the relevant reward semantics and extrapolate beyond the performance of the demonstrator. When combined with deep reinforcement learning, we showed that T-REX achieves better-than-demonstrator performance across a wide range of high-dimensional continuous control tasks and Atari video games as well as outperforming other imitation learning approaches. We also demonstrated that T-REX is robust to modest amounts of ranking noise. However, T-REX assumes access to explicit preference labels, which may be unavailable and difficult to obtain.

We next focused on making reward learning from rankings more applicable to a wider variety of imitation learning tasks where only unlabeled demonstrations are available. We first considered the extension of T-REX to the case where demonstrations come from a learner improving at a task over time. We showed that T-REX naturally extends to this case and can infer the reward function of an improving demonstrator via weakly labeled preferences obtained from timestamps, where later trajectories are assumed to be more preferable to earlier trajectories.

Finally, we considered the most general setting, where demonstrations have no timestamp or preference labels associated with them. We presented a novel imitation learning algorithm, Disturbance-based Reward Extrapolation (D-REX) that automatically generates ranked demonstrations via noise injection and uses these demonstrations to seek to

extrapolate beyond the performance of a suboptimal demonstrator. We empirically evaluated D-REX on a set of simulated robot locomotion and Atari tasks and found that D-REX outperforms state-of-the-art imitation learning techniques and also outperforms the best demonstration in 8 out of 9 tasks. These results provide the first evidence that better-than-demonstrator imitation learning is possible without requiring extra information such as rewards, active supervision, or preference labels. Our results open the door to the application of a variety of ranking and preference-based learning techniques (Cao et al., 2007; Chen et al., 2009) to standard imitation learning domains where only unlabeled demonstrations are available.

The methods proposed in this chapter provide efficient reward inference; however, the main goal of this dissertation is to achieve efficient and safe imitation learning. Chapter 4 demonstrated that utilizing a posterior distribution over reward functions can enable tight, high-confidence bounds on performance; however, the methods in Chapter 4 relied on an MDP solver in the inner-loop. Conversely, the methods proposed in this chapter remove the need for an MDP solver in the inner-loop, but only learn a maximum likelihood estimate of the reward function. In the next chapter we combine the strengths of both approaches by developing a Bayesian version of T-REX that enables high-confidence performance bounds that scale to high-dimensional imitation learning tasks.

Chapter 6

Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences

In Chapter 4 we presented a method for high-confidence policy evaluation for imitation learning that uses Bayesian inverse reinforcement learning (IRL) (Ramachandran and Amir, 2007) to allow an agent to reason about reward uncertainty and policy generalization error (Brown et al., 2018). However, Bayesian IRL is typically intractable for complex problems due to the need to repeatedly solve an MDP in the inner loop, resulting in high computational cost as well as high sample cost if a model is not available. This precludes robust safety and uncertainty analysis for imitation learning in high-dimensional problems or in problems in which a useful model of the MDP is unavailable. Then in Chapter 5 we discussed computationally efficient methods for fast reward inference from preference rankings over small numbers of demonstrations. However, the methods described in Chapter 5 only learn a point estimate of the reward function which does not enable uncertainty analysis or high-confidence performance bounds. In this chapter we present Contribution 6: a deep Bayesian reward inference algorithm that scales to high-dimensional tasks via prefer-

ence rankings over demonstrations.[11]

We seek to address the computational inefficiencies of Chapter 4 via preferences over demonstrations. Preferences over trajectories have been shown to be natural and intuitive for humans to provide (Akrour et al., 2011; Wilson et al., 2012; Sadigh et al., 2017; Christiano et al., 2017; Palan et al., 2019). Furthermore, as we showed in Chapter 5, preferences over demonstrations can allow for better-than-demonstrator performance and efficient reward inference. While preference learning is a common subfield of machine learning, to the best of our knowledge, we are the first to show that preferences over demonstrations enable fast Bayesian reward learning in high-dimensional control tasks as well as enabling efficient high-confidence performance bounds for imitation learning.

In the following sections we propose a novel algorithm, Bayesian Reward Extrapolation (Bayesian REX), that uses a pairwise ranking likelihood to significantly reduce the computational complexity of generating samples from the posterior distribution over reward functions. Bayesian REX scales to high-dimensional imitation learning problems by first pre-training a low-dimensional feature encoding via self-supervised tasks and then leveraging preferences over demonstrations to perform fast Bayesian inference. We demonstrate that Bayesian REX can leverage neural network function approximation to learn useful reward features via self-supervised learning in order to efficiently perform deep Bayesian reward inference from visual demonstrations. For Atari games our approach enables us to generate 100,000 samples from the posterior over reward functions in only 5 minutes using a personal laptop.

Finally, we demonstrate that samples obtained from Bayesian REX can be used to solve the high-confidence policy evaluation problem for imitation learning. We evaluate our method on imitation learning for Atari games and demonstrate that we can efficiently compute high-confidence bounds on policy performance, without requiring samples of the

[11] This work was done in collaboration with Russell Coleman, Scott Niekum, and Ravi Srinivasan and was previously published at ICML 2020 (Brown et al., 2020) and at the NeurIPS 2019 Workshop on Safety and Robustness in Decision Making (Brown and Niekum, 2019a).

reward function. Furthermore, we demonstrate that these high-confidence bounds can be used to accurately rank different evaluation policies according to their risk and performance under the distribution over the unknown ground-truth reward function. Finally, we provide evidence that bounds on uncertainty and risk and may provide a useful tool for detecting reward hacking/gaming (Amodei et al., 2016), a common problem in reward inference from demonstrations (Ibarz et al., 2018) as well as reinforcement learning (Ng et al., 1999; Leike et al., 2017).

6.1 Bayesian Reward Extrapolation

In this and the following sections we describe the main contribution of this chapter: a method for scaling Bayesian reward inference to high-dimensional visual control tasks as a way to efficiently solve the HCPE-IL problem for complex imitation learning tasks. Our first insight is that the main bottleneck for standard Bayesian IRL (see Section 3.3 and Ramachandran and Amir (2007)) is computing the likelihood function

$$P(D|R) = \prod_{(s,a)\in D} \frac{e^{\beta Q^*_R(s,a)}}{\sum_{b\in\mathcal{A}} e^{\beta Q^*_R(s,b)}}, \tag{6.1}$$

which requires optimal Q-values. Thus, to make Bayesian reward inference scale to high-dimensional visual domains, it is necessary to either efficiently approximate optimal Q-values or to formulate a new likelihood. Value-based reinforcement learning focuses on efficiently learning optimal Q-values; however, for visual control tasks such as Atari, RL algorithms can take several hours or even days to train (Mnih et al., 2015; Hessel et al., 2018). This makes MCMC, which requires evaluating large numbers of likelihood ratios, infeasible given the current state-of-the-art in value-based RL. Methods such as transfer learning have great potential to reduce the time needed to calculate Q^*_R for a new proposed reward function R; however, transfer learning is not guaranteed to speed up reinforcement learning (Taylor and Stone, 2009). Thus, we choose to focus on reformulating the likelihood

function as a way to speed up Bayesian reward inference.

An ideal likelihood function requires little computation and minimal interaction with the environment. To accomplish this, we leverage recent work on learning control policies from preferences (Christiano et al., 2017; Palan et al., 2019; Bıyık et al., 2019). In Chapter 6 we presented Trajectory-ranked Reward Extrapolation (T-REX): an efficient reward inference algorithm that uses preferences over demonstrations to transform reward function learning into classification problem via a pairwise ranking loss. T-REX removes the need to repeatedly sample from or partially solve an MDP in the inner loop, allowing it to scale to visual imitation learning domains such as Atari and to extrapolate beyond the performance of the best demonstration. However, T-REX only solves for a point estimate of the reward function. We now discuss how a similar approach based on a pairwise preference likelihood allows for efficient sampling from the posterior distribution over reward functions.

We assume access to a sequence of m trajectories, $D = \{\tau_1, \ldots, \tau_m\}$, along with a set of pairwise preferences over trajectories $\mathcal{P} = \{(i, j) : \tau_i \prec \tau_j\}$. Note that we do not require a total-ordering over trajectories. These preferences may come from a human demonstrator or as proposed in Section 5.3, these preferences could be automatically generated by watching a learner improve at a task (Brown et al., 2019b; Jacq et al., 2019) or via noise injection (Brown et al., 2019a). The benefit of pairwise preferences over trajectories is that we can now leverage a pair-wise ranking loss to compute the likelihood of a set of preferences over demonstrations $\mathcal{P}$, given a parameterized reward function hypothesis R_θ. We use the standard Bradley-Terry model (Bradley and Terry, 1952) to obtain the following pairwise ranking likelihood function, commonly used in learning to rank applications such collaborative filtering (Volkovs and Zemel, 2014):

$$P(\mathcal{P}, D \mid R_\theta) = \prod_{(i,j)\in\mathcal{P}} \frac{e^{\beta R_\theta(\tau_j)}}{e^{\beta R_\theta(\tau_i)} + e^{\beta R_\theta(\tau_j)}}, \tag{6.2}$$

in which $R_\theta(\tau) = \sum_{s\in\tau} R_\theta(s)$ is the predicted return of trajectory τ under the reward

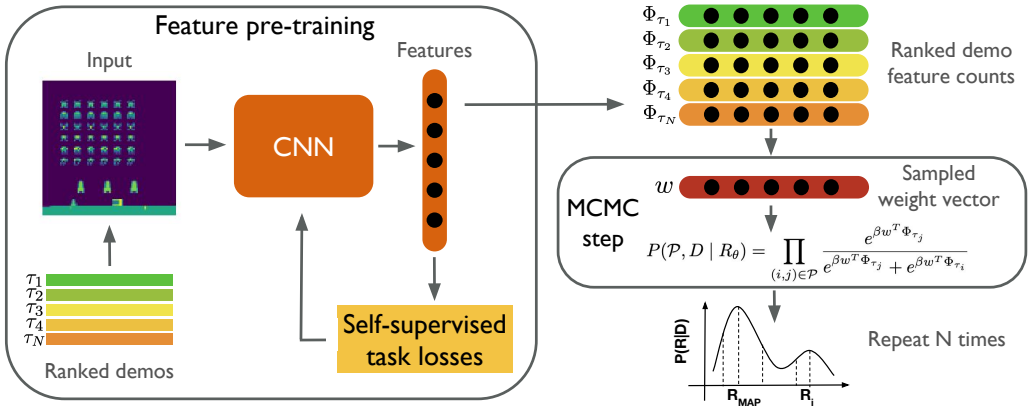


Figure 6.1: Bayesian Reward Extrapolation uses ranked demonstrations to pre-train a low-dimensional state feature embedding $\phi(s)$ via self-supervised losses. After pre-training, the latent embedding function $\phi(s)$ is frozen and the reward function is represented as a linear combination of the learned features: R(s) = $w^T\phi(s)$. MCMC proposal evaluations are computed using an efficient pairwise ranking likelihood that gives the likelihood of the preferences $\mathcal{P}$ over demonstrations D, given a proposal w. By pre-computing the embeddings of the ranked demonstrations, Φ_{τ_i}, MCMC sampling is highly efficient—it does not require access to an MDP solver or data collection during inference.

function R_θ, and β is the inverse temperature parameter that models the confidence in the preference labels. We can then perform Bayesian inference via MCMC to obtain samples from $P(R_\theta \mid D, \mathcal{P}) \propto P(\mathcal{P}, D \mid R_\theta)P(R_\theta)$. We call this approach Bayesian Reward Extrapolation or Bayesian REX.

Note that using the likelihood function defined in Equation (6.2) does not require solving an MDP. In fact, it does not require any rollouts or access to the MDP. All that is required is that we first calculate the return of each trajectory under R_θ and compare the relative predicted returns to the preference labels to determine the likelihood of the demonstrations under the reward hypothesis R_θ. Thus, given preferences over demonstrations, Bayesian REX is significantly more efficient than standard Bayesian IRL. In the following section, we discuss further optimizations that improve the efficiency of Bayesian REX and make it more amenable to our end goal of high-confidence policy evaluation bounds.

6.2 Optimizations

In order to learn rich, complex reward functions, it is desirable to use a deep network to represent the reward function R_θ. While MCMC remains the gold-standard for Bayesian Neural Networks, it is often challenging to scale to deep networks. To make Bayesian REX more efficient and practical, we propose to limit the proposal to only change the last layer of weights in R_θ when generating MCMC proposals—we will discuss pre-training the bottom layers of R_θ in the next section. After pre-training, we freeze all but the last layer of weights and use the activations of the penultimate layer as the latent reward features $\phi(s) \in \mathbb{R}^k$. This allows the reward at a state to be represented as a linear combination of k features: $R_\theta(s) = w^T\phi(s)$. Similar to work by Pradier et al. (2018), operating in the lower-dimensional latent space makes full Bayesian inference tractable.

A second advantage of using a learned linear reward function is that it allows us to efficiently compute likelihood ratios when performing MCMC. Consider the likelihood function in Equation (6.2). If we do not represent R_θ as a linear combination of pre-trained features, and instead let any parameter in R_θ change during each proposal, then for m demonstrations of length T, computing $P(\mathcal{P}, D|R_\theta)$ for a new proposal R_θ requires $O(mT)$ forward passes through the entire network to compute $R_\theta(\tau_i)$. Thus, the complexity of generating N samples from the posterior results is $O(mTN|R_\theta|)$, where $|R_\theta|$ is the number of computations required for a full forward pass through the entire network R_θ. Given that we would like to use a deep network to parameterize R_θ and generate thousands of samples from the posterior distribution over R_θ, this many computations will significantly slow down MCMC proposal evaluation.

If we represent R_θ as a linear combination of pre-trained features, we can reduce this computational cost because

$$R_\theta(\tau) = \sum_{s\in\tau} w^T\phi(s) = w^T \sum_{s\in\tau} \phi(s) = w^T\Phi_\tau. \tag{6.3}$$

Thus, we can pre-compute and cache $\Phi_{\tau_i} = \sum_{s \in \tau_i} \phi(s)$ for $i = 1, \ldots, m$ and the likelihood becomes

$$P(\mathcal{P}, D \mid R_\theta) = \prod_{(i,j) \in \mathcal{P}} \frac{e^{\beta w^T \Phi_{\tau_j}}}{e^{\beta w^T \Phi_{\tau_j}} + e^{\beta w^T \Phi_{\tau_i}}}. \tag{6.4}$$

Note that demonstrations only need to be passed through the reward network once to compute Φ_{τ_i} since the pre-trained embedding remains constant during MCMC proposal generation. This results in an initial $O(mT)$ passes through all but the last layer of R_θ to obtain Φ_{τ_i}, for $i = 1, \ldots, m$, and then only $O(mk)$ multiplications per proposal evaluation thereafter—each proposal requires that we compute $w^T \Phi_{\tau_i}$ for $i = 1, \ldots, m$ and $\Phi_{\tau_i} \in \mathbb{R}^k$. Thus, when using feature pre-training, the total complexity is only $O(mT|R_\theta| + mkN)$ to generate N samples via MCMC. This reduction in the complexity of MCMC from $O(mTN|R_\theta|)$ to $O(mT|R_\theta| + mkN)$ results in significant and practical computational savings because (1) we want to make N and R_θ large and (2) the number of demonstrations, m, and the size of the latent embedding, k, are typically several orders of magnitude smaller than N and $|R_\theta|$.

A third, and critical advantage of using a learned linear reward function is that it makes solving the HCPE-IL problem discussed in Chapter 4 tractable. Performing a single policy evaluation is a non-trivial task (Sutton et al., 2000) and even in tabular settings has complexity $O(|S|^3)$ in which $|S|$ is the size of the state-space (Littman et al., 1995). Because we are in an imitation learning setting, we would like to be able to efficiently evaluate any given policy across the posterior distribution over reward functions found via Bayesian REX. Given a posterior distribution over N reward function hypotheses we would need to solve N policy evaluations. However, as noted in Chapter 4, the value function of a policy under a reward function $R(s) = w^T \phi(s)$ can be written as

$$V_R^\pi = \mathbb{E}_\pi[\sum_{t=0}^{T} R(s_t)] = w^T \mathbb{E}_\pi[\sum_{t=0}^{T} \phi(s_t)] = w^T \Phi_\pi, \tag{6.5}$$

in which we assume a finite horizon MDP with horizon T and in which Φ_π are the expected

feature counts (Abbeel and Ng, 2004; Barreto et al., 2017) of π. Thus, given any evaluation policy π_{eval}, we only need to solve one policy evaluation problem to compute Φ_{eval}. We can then compute the expected value of π_{eval} over the entire posterior distribution of reward functions via a single matrix vector multiplication $W\Phi_{\pi_{\text{eval}}}$, where W is an N-by-k matrix with each row corresponding to a single reward function weight hypothesis w^T. This significantly reduces the complexity of policy evaluation over the reward function posterior distribution from $O(N|S|^3)$ to $O(|S|^3 + Nk)$.

When we refer to Bayesian REX we will refer to the optimized version described in this section. Our approach is summarized in Algorithm 4. Running Bayesian REX with 144 preference labels to generate 100,000 reward hypothesis for Atari imitation learning tasks takes approximately 5 minutes on a Dell Inspiron 5577 personal laptop with an Intel i7-7700 processor without using the GPU. In comparison, using standard Bayesian IRL to generate *one sample* from the posterior takes 10+ hours of training for a parallelized PPO reinforcement learning agent (Dhariwal et al., 2017) on an NVIDIA TITAN V GPU.

6.3 Pre-Training Reward Function Features

The previous section presupposed access to a pre-trained latent embedding function $\phi : S \rightarrow \mathbb{R}^k$. We now discuss our pre-training process. Because we are interested in imitation learning problems, we need to be able to train $\phi(s)$ from the demonstrations without access to the ground-truth reward function. One potential method is to train R_θ using the pairwise ranking likelihood function in Equation (6.2) and then freeze all but the last layer of weights; however, the learned embedding may overfit to the limited number of preferences over demonstrations and fail to capture features relevant to the ground-truth reward function. Thus, we supplement the pairwise ranking objective with auxiliary objectives that can be optimized in a self-supervised fashion using data from the demonstrations.

We use the following self-supervised tasks to pre-train R_θ: (1) Learn an inverse dynamics model that uses embeddings $\phi(s_t)$ and $\phi(s_{t+1})$ to predict the corresponding action

Algorithm 4 Bayesian REX: Bayesian Reward Extrapolation

Input: demonstrations D, pairwise preferences $\mathcal{P}$, MCMC proposal width σ, number of proposals to generate N, deep network architecture R_θ, and prior $P(w)$.
pre-train R_θ using auxiliary tasks (see Section 5.2).
Freeze all but last layer, w, of R_θ.
$\phi(s) :=$ activations of the penultimate layer of R_θ.
Pre-compute and cache $\Phi_\tau = \sum_{s \in \tau} \phi(s)$ for all $\tau \in D$.
Initialize w randomly.
Chain[0] $\leftarrow w$
Compute $P(\mathcal{P}, D|w)P(w)$ using Equation (6.4)
for $i \leftarrow 1$ to N **do**
    $\tilde{w} \leftarrow \texttt{normalize}(\mathcal{N}(w, \sigma))$
    Compute $P(\mathcal{P}, D|\tilde{w})P(\tilde{w})$ using Equation (6.4)
    $u \leftarrow \texttt{Uniform}(0, 1)$
    if $u < \dfrac{P(\mathcal{P}, D|\tilde{w})P(\tilde{w})}{P(\mathcal{P}, D|w)P(w)}$ **then**
        Chain[i] $\leftarrow \tilde{w}$
        $w \leftarrow \tilde{w}$
    else
        Chain[i] $\leftarrow w$
return Chain

a_t (Torabi et al., 2018a; Hanna and Stone, 2017), (2) Learn a forward dynamics model that predicts s_{t+1} from $\phi(s_t)$ and a_t (Oh et al., 2015; Thananjeyan et al., 2019), (3) Learn an embedding $\phi(s)$ that predicts the temporal distance between two randomly chosen states from the same demonstration (Aytar et al., 2018), and (4) Train a variational pixel-to-pixel autoencoder in which $\phi(s)$ is the learned latent encoding (Makhzani and Frey, 2017; Doersch, 2016). Table 6.1 summarizes the self-supervised tasks used to train $\phi(s)$ and Figure 6.2 summarizes our network architecture.

There are many possibilities for pre-training $\phi(s)$; however, we found that each objective described above encourages the embedding to encode different features. For example, an accurate inverse dynamics model can be learned by only attending to the movement of the agent. Learning forward dynamics supplements this by requiring $\phi(s)$ to encode information about short-term changes to the environment. Learning to predict the temporal

Table 6.1: Self-supervised learning objectives used to pre-train $\phi(s)$.

Inverse Dynamics	$f_{\text{ID}}(\phi(s_t), \phi(s_{t+1})) \rightarrow a_t$
Forward Dynamics	$f_{\text{FD}}(\phi(s_t), a_t) \rightarrow s_{t+1}$
Temporal Distance	$f_{\text{TD}}(\phi(s_t), \phi(s_{t+x}) \rightarrow x$
Variational Autoencoder	$f_A(\phi(s_t)) \rightarrow s_t$

distance between states in a trajectory forces $\phi(s)$ to encode long-term progress. Finally, the autoencoder loss acts as a regularizer to the other losses as it seeks to embed all aspects of the state (see Appendix E.1 for details). The full Bayesian REX pipeline for generating samples from the posterior over reward functions is summarized in Figure 6.1.

6.4 HCPE-IL via Bayesian REX

We now discuss how to use Bayesian REX to find an efficient solution to the high-confidence policy evaluation for imitation learning (HCPE-IL) problem. While this problem was presented earlier in Section 4.1, in this chapter we seek a lower bound rather than an upper bound. Thus, we rephrase this problem for completeness.

6.4.1 High-Confidence Policy Evaluation for Imitation Learning

We assume access to an MDP\R, an evaluation policy π_{eval}, a set of demonstrations, $D = \{\tau_1, \ldots, \tau_m\}$, in which τ_i is either a complete or partial trajectory comprised of states or state-action pairs, a confidence level δ, and performance metric $g : \Pi \times \mathcal{R} \rightarrow \mathbb{R}$, in which $\mathcal{R}$ denotes the space of reward functions and Π is the space of all policies.

The *High-Confidence Policy Evaluation problem for Imitation Learning* (HCPE-IL) is to find a high-confidence lower bound $\hat{g} : \Pi \times \mathcal{D} \rightarrow \mathbb{R}$ such that

$$\Pr(g(\pi_{\text{eval}}, R^*) \geq \hat{g}(\pi_{\text{eval}}, D)) \geq 1 - \delta, \tag{6.6}$$

in which R^* denotes the demonstrator's true reward function and $\mathcal{D}$ denotes the space of

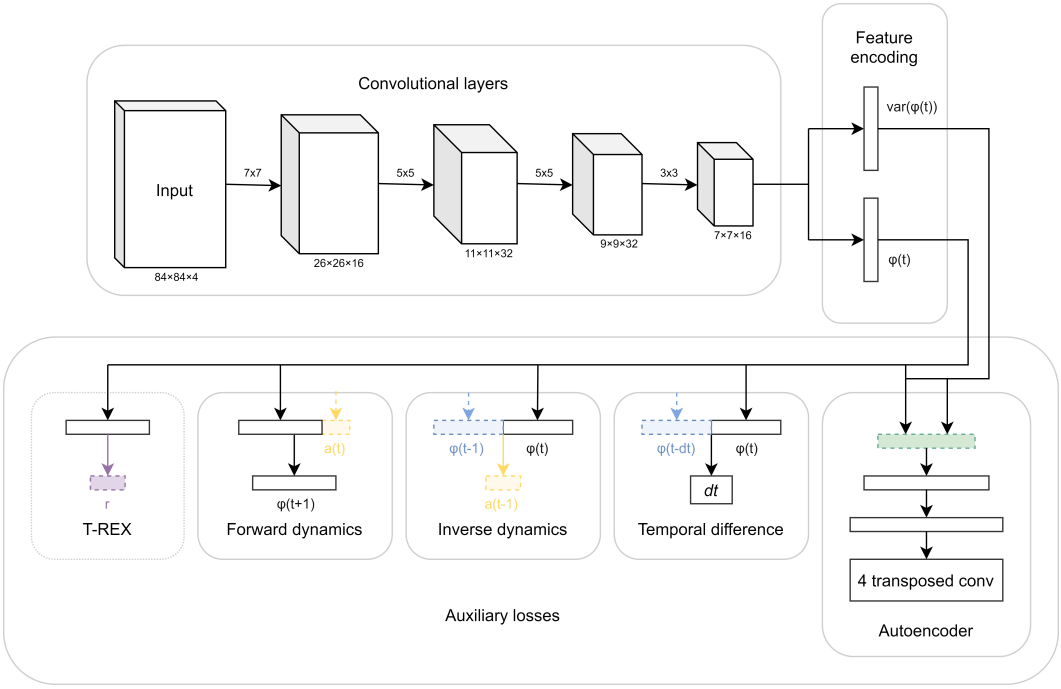


Figure 6.2: Diagram of the network architecture used when training feature encoding $\phi(s)$ with self-supervised and T-REX losses. Yellow denotes actions, blue denotes feature encodings sampled from elsewhere in a demonstration trajectory, and green denotes random samples for the variational autoencoder.

all possible demonstration sets. HCPE-IL takes as input an evaluation policy π_{eval}, a set of demonstrations D, and a performance metric, g, which evaluates a policy under a reward function. The goal of HCPE-IL is to return a high-confidence lower bound $\hat{g}$ on the performance statistic $g(\pi_{\text{eval}}, R^*)$.

6.4.2 Computation Details

Given samples from the distribution $P(w|D, \mathcal{P})$, in which $R(s) = w^T\phi(s)$, we compute the posterior distribution over any performance statistic $g(\pi_{\text{eval}}, R^*)$ as follows. For each sampled weight vector w produced by Bayesian REX, we compute $g(\pi_{\text{eval}}, w)$. This results in a sample from the posterior distribution $P(g(\pi_{\text{eval}}, R)|\mathcal{P}, D)$, i.e., the posterior distribution over performance metric g, conditioned on D and $\mathcal{P}$. We then compute a $(1-\delta)$ confidence

lower bound, $\hat{g}(\pi_{\text{eval}}, D)$, by finding the δ-quantile of $g(\pi_{\text{eval}}, w)$ for $w \sim P(w|\mathcal{P}, D)$.

While there are many potential performance statistics g, in this chapter we focus on bounding the expected value of the evaluation policy, i.e., $g(\pi_{\text{eval}}, R^*) = V_{R^*}^{\pi_{\text{eval}}} = w^{*T}\Phi_{\pi_{\text{eval}}}$. To compute a $1-\delta$ confidence bound on $V_{R^*}^{\pi_{\text{eval}}}$, we take full advantage of the learned linear reward representation to efficiently calculate the posterior distribution over policy returns given preferences and demonstrations. The posterior distribution over returns is calculated via a matrix vector product, $W\Phi_{\pi_{\text{eval}}}$, in which each row of W is a sample, w, from the MCMC chain and π_{eval} is the evaluation policy. We then sort the resulting vector and select the δ-quantile lowest value. This results in a $1-\delta$ confidence lower bound on $V_{R^*}^{\pi_{\text{eval}}}$ and corresponds to the δ-Value at Risk (VaR) over $V_{R}^{\pi_{\text{eval}}} \sim P(R|\mathcal{P}, D)$ (Jorion, 1997).

6.5 Experimental Comparison of Bayesian IRL vs. Bayesian REX

Bayesian IRL (Ramachandran and Amir, 2007) does not scale to high-dimensional tasks due to the requirement of repeatedly solving for an MDP in the inner loop. In this section we focus on low-dimensional problems where it is tractable to repeatedly solve an MDP. We compare the performance of Bayesian IRL with Bayesian REX when performing reward inference. Because both algorithms make very different assumptions, we compare their performance across three different tasks. The first task attempts to give both algorithms the demonstrations they were designed for. The second evaluation focuses on the case where all demonstrations are optimal and is designed to put Bayesian IRL at a disadvantage. The third evaluation focuses on the case where all demonstrations are optimal and is designed to put Bayesian REX at a disadvantage. Note that we focus on sample efficiency rather than computational efficiency as Bayesian IRL is significantly slower than Bayesian REX as it requires repeatedly solving an MDP, whereas Bayesian REX requires no access to an MDP

during reward inference.

All experiments were performed using 6x6 gridworlds with 4 binary features placed randomly in the environment. The ground-truth reward functions are sampled uniformly from the L1-ball (Brown and Niekum, 2018). The agent has 4 actions in the cardinal directions and self transitions if it attempts to move off the grid. Transitions are deterministic, $\gamma = 0.9$, and there are no terminal states. We perform evaluations over 100 random gridworlds for varying numbers of demonstrations. Each demonstration is truncated to a horizon of 20. We use $\beta = 50$ for both Bayesian IRL and Bayesian REX and we remove duplicates from demonstrations. After performing MCMC we used a 10% burn-in period for both algorithms and only used every 5th sample after the burn-in when computing the mean reward under the posterior. We then optimized a policy under the mean reward from the Bayesian IRL posterior and under the mean reward from the Bayesian REX posterior. We then compare the average policy loss for each algorithm when compared with optimal performance under the ground-truth reward function.

6.5.1 Ranked Suboptimal vs. Optimal Demonstrations

We first compare Bayesian IRL when it is given optimal demonstrations with Bayesian REX when it receives suboptimal demonstrations. We give each algorithm the demonstrations best suited for its assumptions while keeping the number of demonstrations equal and using the same starting states for each algorithm. To generate suboptimal demonstrations we simply use random rollouts and then rank them according to the ground-truth reward function.

Table 6.2 shows that, when given a sufficient number of suboptimal ranked demonstrations (> 5), Bayesian REX performs on par or slightly better than full Bayesian IRL when given the same number of optimal demonstrations starting from the same states as the suboptimal demonstrations. This result is encouraging as it shows that not only is Bayesian REX much more computationally efficient, but it's sample efficiency compara-

Table 6.2: Ranked Suboptimal vs. Optimal Demos: Average policy loss over 100 random 6x6 grid worlds with four binary features.

	2	5	10	20	30
Bayesian IRL	**0.044**	**0.033**	0.020	0.009	**0.006**
Bayesian REX	1.779	0.421	**0.019**	**0.006**	**0.006**

Table 6.3: Ranked Suboptimal Demos: Average policy loss for Bayesian IRL versus Bayesian REX over 100 random 6x6 grid worlds with 4 binary features.

	2	5	10	20	30
Bayesian IRL	3.512	3.319	2.791	3.078	3.158
Bayesian REX	**1.796**	**0.393**	**0.026**	**0.006**	**0.006**

ble to Bayesian IRL as long as there are a sufficient number of ranked demonstrations. Note that 2 ranked demonstrations induces only a single constraint on the reward function so it is not surprising that it performs much worse than running full Bayesian IRL with all the counterfactuals afforded by running an MDP solver in the inner-loop.

6.5.2 Only Ranked Suboptimal Demonstrations

For the next experiment we consider what happens when Bayesian IRL recieves suboptimal ranked demonstrations. Table 6.3 shows that B-REX always significantly outperforms Bayesian IRL when both algorithms receive suboptimal ranked demonstrations. To achieve a fairer comparison, we also compared Bayesian REX with a Bayesian IRL algorithm designed to learn from both good and bad demonstrations (Cui and Niekum, 2018). We labeled the top $x\%$ ranked demonstrations as good and bottom $x\%$ ranked as bad. Table 6.4 shows that leveraging the ranking significantly improves the performance of Bayesian IRL, but Bayesian REX still performed significantly better across all x.

Table 6.4: Ranked Suboptimal Demos: Average policy loss for Bayesian REX and Bayesian IRL using the method proposed by (Cui and Niekum, 2018)* which makes use of good and bad demonstrations. We used the top $x\%$ of the ranked demos as good and bottom $x\%$ as bad. Results are averaged over 100 random 6x6 grid worlds with four binary features.

	Top/bottom percent of 20 ranked demos			
	x=5%	x=10%	x=25%	x=50%
Bayesian IRL(x)*	1.283	0.956	1.065	2.096
Bayesian REX			**0.006**	

Table 6.5: Ranked Suboptimal Demos: Average policy loss for Bayesian IRL versus Bayesian REX over 100 random 6x6 grid worlds with four binary features.

	2	5	10	20	30
Bayesian IRL	**0.045**	**0.034**	**0.018**	**0.009**	**0.006**
Bayesian REX	0.051	0.045	0.040	0.034	0.034

6.5.3 Only Optimal Demonstrations

Finally, we compared Bayesian REX with Bayesian IRL when both algorithms are given optimal demonstrations. As an attempt to use Bayesian REX with only optimal demonstrations, we followed prior work (Brown et al., 2019a) and auto-generated pairwise preferences using uniform random rollouts that are labeled as less preferred than the demonstrations. Table 6.5 shows that Bayesian IRL outperforms Bayesian REX. This demonstrates the value of giving a variety of ranked trajectories to Bayesian REX. For small numbers of optimal demonstrations (≤ 5) we found that Bayesian REX leveraged the self-supervised rankings to only perform slightly worse than full Bayesian IRL. This result is encouraging since it is possible that a more sophisticated method for auto-generating suboptimal demonstrations and rankings could be used to further improve the performance of Bayesian REX even when demonstrations are not ranked (Brown et al., 2019a).

The results above demonstrate that if a very small number of unlabeled near-optimal demonstrations are available, then classical Bayesian IRL is the natural choice for perform-

ing reward inference. However, if any of these assumptions are not true, then Bayesian REX is a competitive and often superior alternative for performing Bayesian reward inference. Also implicit in the above results is the assumption that a highly tractable MDP solver is available for performing Bayesian IRL. If this is not the case, then Bayesian IRL is infeasible and Bayesian REX is the natural choice for Bayesian reward inference.

6.6 Visual Imitation Learning Results

We next tested the imitation learning performance of Bayesian REX for high-dimensional problems where classical Bayesian reward inference is infeasible. We pre-trained a 64 dimensional latent state embedding $\phi(s)$ using the self-supervised losses shown in Table 6.1 and the T-REX pairwise preference loss. We found via ablation studies that combining the T-REX loss with the self-supervised losses resulted in better performance than training only with the T-REX loss or only with the self-supervised losses (see Appendix E.1 E.1 for details). We then used Bayesian REX to generate 200,000 samples from the posterior $P(R|D, \mathcal{P})$. We then took the MAP and mean reward function estimates from the posterior and optimized a policy using Proximal Policy Optimization (PPO) (Schulman et al., 2017) (see Appendix E.1 for details).

To test whether Bayesian REX scales to complex imitation learning tasks we selected five Atari games from the Arcade Learning Environment (Bellemare et al., 2013). We do not give the RL agent access to the ground-truth reward signal and mask the game scores and number of lives in the demonstrations. Table 6.6 shows the imitation learning performance of Bayesian REX. We also compare against the results reported by (Brown et al., 2019b) for T-REX, and GAIL (Ho and Ermon, 2016) and use the same 12 suboptimal demonstrations used by Brown et al. (2019b) to train Bayesian REX (see Appendix E.1 for details).

Table 6.6 shows that Bayesian REX is able to utilize preferences over demonstrations to infer an accurate reward function that enables better-than-demonstrator perfor-

Table 6.6: Ground-truth average scores when optimizing the mean and MAP rewards found using Bayesian REX. We also compare against the performance of T-REX (Brown et al., 2019b) and GAIL (Ho and Ermon, 2016). Bayesian REX and T-REX are each given 12 demonstrations with ground-truth pairwise preferences. GAIL cannot learn from preferences so it is given 10 demonstrations comparable to the best demonstration given to the other algorithms. The average performance for each IRL algorithm is the average over 30 rollouts.

	Ranked Demonstrations		B-REX Mean	B-REX MAP	T-REX	GAIL
Game	Best	Avg	Avg	Avg	Avg	Avg
Beam Rider	1332	686.0	5,504.7	**5,870.3**	3,335.7	355.5
Breakout	32	14.5	390.7	**393.1**	221.3	0.28
Enduro	84	39.8	487.7	135.0	**586.8**	0.28
Seaquest	600	373.3	734.7	606.0	**747.3**	0.0
Space Invaders	600	332.9	**1,118.8**	961.3	1,032.5	370.2

mance. The average ground-truth return for Bayesian REX surpasses the performance of the best demonstration across all 5 games. In comparison, GAIL seeks to match the demonstrator's state-action distributions which makes imitation learning difficult when demonstrations are suboptimal and noisy. In addition to providing uncertainty information, Bayesian REX remains competitive with T-REX (which only finds a maximum likelihood estimate of the reward function) and achieves better performance on 3 out of 5 games.

6.7 High-Confidence Policy Evaluation Results

Next, we ran an experiment to validate whether the posterior distribution generated by Bayesian REX can be used to solve the HCPE-IL problem described in Section 6.4.1. We first evaluated four different evaluation policies, $A \prec B \prec C \prec D$, created by partially training a PPO agent on the ground-truth reward function and checkpointing the policy at various stages of learning. We ran Bayesian REX to generate 200,000 samples from $P(R|\mathcal{P}, D)$. To address some of the ill-posedness of IRL, we normalize the weights w such that $\|w\|_2 = 1$. With rewards and returns all on the same scale we can compare the relative

Table 6.7: Beam Rider policy evaluation bounds compared with ground-truth game scores. Policies A-D correspond to evaluation policies of varying quality obtained by checkpointing an RL agent during training. The No-Op policy seeks to hack the learned reward by always playing the no-op action, resulting in very long trajectories with high mean predicted performance but a very negative 95%-confidence (0.05-VaR) lower bound on expected return.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	17.1	7.9	480.6	1372.6
B	22.7	11.9	703.4	1,412.8
C	45.5	24.9	1828.5	2,389.9
D	57.6	31.5	2586.7	2,965.0
No-Op	102.5	-1557.1	0.0	99,994.0

performance of several different evaluation policies when evaluated over the posterior.

The results for Beam Rider are shown in Table 6.7. We show results for partially trained RL policies A-D. We found that the ground-truth returns for the checkpoints were highly correlated with the mean and 0.05-VaR (5th percentile policy return) returns under the posterior. However, we also noticed that the trajectory length was also highly correlated with the ground-truth reward. If the reward function learned via IRL gives a small positive reward at every timestep, then long polices that do the wrong thing may look good under the posterior. To test this hypothesis we used a No-Op policy that seeks to exploit the learned reward function by not taking any actions. This allows the agent to live until the Atari emulator times out after 99,994 steps.

Table 6.7 shows that while the No-Op policy has a high expected return over the chain, looking at the 0.05-VaR shows that the No-Op policy has high risk under the distribution, much lower than evaluation policy A. Our results demonstrate that reasoning about probabilistic worst-case performance may be one potential way to detect policies that exhibit so-called reward hacking (Amodei et al., 2016) or that have overfit to certain features in the demonstrations that are correlated with the intent of the demonstrations, but do not lead to desired behavior, a common problem in imitation learning (Ibarz et al., 2018; de Haan

Table 6.8: Breakout policy evaluation bounds compared with ground-truth game scores. Top Half: No-Op never releases the ball, resulting in high mean predicted performance but a low 95%-confidence bound (0.05-VaR). The MAP policy has even higher risk but also high expected return. Bottom Half: After rerunning MCMC with a ranked trajectory from both the MAP and No-Op policies, the posterior distribution matches the true preferences.

Risk profiles given initial preferences				
	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	1.5	0.5	1.9	202.7
B	6.3	3.7	15.8	608.4
C	10.6	5.8	27.7	849.3
D	13.9	6.2	41.2	1020.8
MAP	98.2	-370.2	401.0	8780.0
No-Op	41.2	1.0	0.0	7000.0
Risk profiles after rankings w.r.t. MAP and No-Op				
A	0.7	0.3	1.9	202.7
B	8.7	5.5	15.8	608.4
C	18.3	12.1	27.7	849.3
D	26.3	17.1	41.2	1020.8
MAP	606.8	289.1	401.0	8780.0
No-Op	-5.0	-13.5	0.0	7000.0

et al., 2019).

Table 6.8 contains policy evaluation results for the game Breakout. The top half of the table shows the mean return and 95%-confidence lower bound on the expected return under the reward function posterior for four evaluation policies as well as the MAP policy found via Bayesian IRL and a No-Op policy that never chooses to release the ball. Both the MAP and No-Op policies have high expected returns under the reward function posterior, but also have high risk (low 0.05-VaR). The MAP policy has much higher risk than the No-Op policy, despite good true performance. One likely reason is that, as shown in Table 6.6, the best demonstrations given to Bayesian REX only achieved a game score of 32. Thus, the MAP policy represents an out of distribution sample and thus has potentially high risk,

since Bayesian REX was not trained on policies that hit any of the top layers of bricks. The ranked demonstrations do not give enough evidence to eliminate the possibility that only lower layers of bricks should be hit.

To test this hypothesis, we added two new ranked demonstrations, a single rollout from the MAP policy and a single rollout from the No-Op policy to the original set of 12 ranked demonstrations and performed MCMC with these new preferences. As the bottom of Table 6.8 shows, adding two more ranked demonstrations results in a significant change in the risk profiles of the MAP and No-Op policy—the No-Op policy is now correctly predicted to have high risk and low expected returns and the MAP policy now has a much higher 95%-confidence lower bound on performance.

To further test Bayesian REX on different learned policies we took a policy trained with RL on the ground truth reward function for Beam Rider, the MAP policy learned via Bayesian REX for Beam Rider, and a policy trained with an earlier version of Bayesian REX (trained without all of the auxiliary losses) that learned a novel reward hack where the policy repeatedly presses left and then right, enabling the agent's ship to stay in between two of the firing lanes of the enemies. The imitation learning reward hack allows the agent to live for a very long time. We took a 2000 step prefix of each policy and evaluated the expected and 5th percentile worst-case predicted returns for each policy. We found that Bayesian REX is able to accurately predict that the reward hacking policy is worse than both the RL policy and the policy optimizing the Bayesian REX reward. However, we found that the Bayesian REX policy, while not performing as well as the RL policy, was given higher expected return and a higher lower bound on performance than the RL policy. Results are shown in Table 6.9.

Table 6.9: Beamrider policy evaluation for an RL policy trained on ground truth reward, an imitation learning policy, and a reward hacking policy that exploits a game hack to live for a long time by moving quickly back and forth.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
RL	36.7	19.5	2135.2	2000
Bayesian REX	68.1	38.1	649.4	2000
Reward Hacking	28.8	10.2	2.2	2000

6.8 High-Confidence Performance Bounds on Human Trajectories

To investigate whether Bayesian REX is able to correctly rank human demonstrations, one of the authors provided demonstrations of a variety of different behaviors and then we took the latent embeddings of the demonstrations and used the posterior distribution to find high-confidence performance bounds for these different rollouts.

6.8.1 Beam Rider

We generated four human demonstrations: (1) *good*, a good demonstration that plays the game well, (2) *bad*, a bad demonstration that seeks to play the game but does a poor job, (3) *suicidal*, a demonstration that does not shoot enemies and seeks enemy bullets, and (4) *adversarial* a demonstration that pretends to play the game by moving and shooting as much as possibly but tries to avoid actually shooting enemies. The results of high-confidence policy evaluation are shown in Table 6.10. The high-confidence bounds and average performance over the posterior correctly rank the behaviors. This provides evidence that the learned linear reward correctly rewards actually destroying aliens and avoiding getting shot, rather than just flying around and shooting.

Table 6.10: Beam Rider evaluation of a variety of human demonstrations.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	12.4	5.8	1092	1000.0
bad	10.7	4.5	396	1000.0
suicidal	6.6	0.8	0	1000.0
adversarial	8.4	2.4	176	1000.0

6.8.2 Space Invaders

For Space Invaders we demonstrated an even wider variety of behaviors to see how Bayesian REX would rank their relative performance. We evaluated the following policies: (1) *good*, a demonstration that attempts to play the game as well as possible, (2) *every other*, a demonstration that only shoots aliens in the 2nd and 4th columns, (3) *flee*, a demonstration that did not shoot aliens, but tried to always be moving while avoiding enemy lasers, (4) *hide*, a demonstration that does not shoot and hides behind on of the barriers to avoid enemy bullets, (5) *suicidal*, a policy that seeks enemy bullets while not shooting, (6) *shoot shelters*, a demonstration that tries to destroy its own shelters by shooting at them, (7) *hold 'n fire*, a demonstration where the player rapidly fires but does not move to avoid enemy lasers, and (8) *miss*, a demonstration where the demonstrator tries to fire but not hit any aliens while avoiding enemy lasers.

Table 6.11 shows the results of evaluating the different demonstrations. The good demonstration is clearly the best performing demonstration in terms of mean performance and 95%-confidence lower bound on performance and the suicidal policy is correctly given the lowest performance lower bound. However, we found that the length of the demonstration appears to have a strong effect on the predicted performance for Space Invaders. Demonstrations such as hide and miss are able to live for a longer time than policies that actually hit aliens. This results in them having higher 0.05-quantile worst-case predicted performance and higher mean performance.

Table 6.11: Space Invaders evaluation of a variety of human demonstrations.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	198.3	89.2	515	1225.0
every other	56.2	25.9	315	728.0
hold 'n fire	44.3	18.6	210	638.0
shoot shelters	47.0	20.6	80	712.0
flee	45.1	19.8	0	722.0
hide	83.0	39.0	0	938.0
miss	66.0	29.9	0	867.0
suicidal	0.5	-13.2	0	266.0

Table 6.12: Space Invaders evaluation of a variety of human demonstrations when considering only the first 6000 steps.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	47.8	22.8	515	600.0
every other	34.6	15.0	315	600.0
hold 'n fire	40.9	17.1	210	600.0
shoot shelters	33.0	13.3	80	600.0
flee	31.3	11.9	0	600.0
hide	32.4	13.8	0	600.0
miss	30.0	11.3	0	600.0

To study this further we looked at only the first 600 timesteps of each policy, to remove any confounding by the length of the trajectory. The results are shown in Table 6.12. With a fixed length demonstration, Bayesian REX is able to correctly rank *good*, *every other*, and *hold 'n fire* as the best demonstrations, despite evaluation policies that are deceptive.

6.8.3 Enduro

For Enduro we tested four different human demonstrations: (1) *good* a demonstration that seeks to play the game well, (2) *periodic* a demonstration that alternates between speeding

Table 6.13: Enduro evaluation of a variety of human demonstrations.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	246.7	-113.2	177	3325.0
periodic	230.0	-130.4	44	3325.0
neutral	190.8	-160.6	0	3325.0
ram	148.4	-214.3	0	3325.0

up and passing cars and then slowing down and being passed, (3) *neutral* a demonstration that stays right next to the last car in the race and doesn't try to pass or get passed, and (4) *ram* a demonstration that tries to ram into as many cars while going fast. Table 6.13 shows that Bayesian REX is able to accurately predict the performance and risk of each of these demonstrations and gives the highest (lowest 0.05-VaR) risk to the *ram* demonstration and the least risk to the *good* demonstration.

6.9 Summary

Bayesian reasoning is a powerful tool when dealing with uncertainty and risk; however, existing Bayesian reward learning algorithms often require solving an MDP in the inner loop, rendering them intractable for complex problems in which solving an MDP may take several hours or even days. In this chapter we proposed a novel deep learning algorithm, Bayesian Reward Extrapolation (Bayesian REX), that leverages preference labels over demonstrations to make Bayesian reward inference tractable for high-dimensional visual imitation learning tasks. Bayesian REX extends T-REX (Chapter 5) to allow for fast Bayesian inference and can sample tens of thousands of reward functions from the posterior in a matter of minutes using a consumer laptop. We tested our approach on five Atari imitation learning tasks and demonstrated Bayesian REX achieves state-of-the-art performance in 3 out of 5 games. Bayesian REX also extends the results of Chapter 4 by enabling efficient high-confidence performance bounds for a wide range of evaluation policies and evaluation

trajectories in high-dimensional, visual imitation learning tasks. Furthermore, we demonstrated that these high-confidence bounds allow accurate comparison of different evaluation policies and provide a potential way to detect reward hacking and value misalignment.

In Chapter 4 and the current chapter, we have focused on the problem of high-confidence policy evaluation. While policy evaluation is an important part of safe imitation learning, we have not addressed in detail the question of what an agent should do if it determines that, with high-probability, it has poor performance. Nor have we addressed how an agent should go about optimizing its policy to have good performance with high-confidence in the first place. In the next two chapters we consider the problems of risk-aware policy optimization. In Chapter 7 we propose an approach for using a posterior distribution to directly optimize a robust policy with respect to a specified risk-return trade-off. In Chapter 8 we examine the problem of risk-aware active inverse reinforcement learning, where an agent can ask for additional help from a demonstrator in order to optimize a robust policy via repeated interactions with a demonstrator.

Chapter 7

Bayesian Robust Optimization for Imitation Learning

In previous chapters we have considered the problem of high-confidence policy evaluation (chapters 4 and 6) and risk-neutral imitation learning (Chapter 5). We now turn our attention to the problem of robust imitation learning via high-confidence policy optimization. This chapter discusses Contribution 7 of this dissertation: an algorithm for Bayesian robust imitation learning that optimizes a policy to balance risk and expected return given a posterior distribution over reward functions.[12]

One of the main challenges in imitation learning is determining what action an agent should take when outside the states contained in the demonstrations. Inverse reinforcement learning (IRL) (Ng and Russell, 2000) is an approach to imitation learning in which the learning agent seeks to recover the reward function of the demonstrator. Learning a parameterized reward function provides a compact representation of the demonstrator's values and enables generalization to new states unseen in the demonstrations via policy optimization. However, IRL approaches still result in uncertainty over the true reward function and this uncertainty can have negative consequences if the learning agent infers a reward func-

[12]This chapter contains work that was done in collaboration with Marek Petrik and Scott Niekum.

tion that leads to learn an incorrect policy. In this chapter we propose that an imitation learning agent should learn a policy that is robust with respect to its uncertainty over the true objective of a task, but also be able to effectively trade-off epistemic risk with expected return.

For example, consider two scenarios: (1) an autonomous car detects a novel object lying in the road ahead of the car and (2) a domestic service robot tasked with vacuuming encounters a pattern on the floor it has never seen before. The first example concerns autonomous driving where the car's decisions have potentially catastrophic consequences. Thus, the car should treat the novel object as a hazard and either slow down or safely change lanes to avoid running into it. In the second example, vacuuming the floors of a house has certain risks, but the consequences of optimizing the wrong reward function are arguably much less significant. Thus, when the vacuuming robot encounters a novel floor pattern it does not need to worry as much about negative side-effects.

Risk-averse optimization, especially in financial domains, has a long history of seeking to address the trade-off between risk and return using measures of risk such as variance (Markowitz and Todd, 2000; Brown, 2011), value at risk (Jorion, 1997) and conditional value at risk (Rockafellar and Uryasev, 2000). This work has been extended to risk-averse optimization in Markov decision processes (Ghavamzadeh et al., 2016; Chow et al., 2015; Petrik and Subramanian, 2012) and in the context of reinforcement learning (Garcıa and Fernández, 2015; Tamar et al., 2015; Tang et al., 2020), where the transition dynamics and reward function are not known. However, there has only been limited work in applying techniques for trading off risk and return in the domain of imitation learning. In chapters 4 and 6 we discussed methods for bounding the value at risk of a policy in the imitation learning setting; however, directly optimizing a policy for value at risk is NP-hard (Delage and Mannor, 2010). Lacotte et al. (2019) and Majumdar et al. (2017) assume that risk-sensitive trajectories are available from a safe demonstrator and seek to optimize a policy that matches the risk-profile of this expert. In contrast, our approach directly optimizes a

policy that balances expected return and conditional value at risk (Rockafellar and Uryasev, 2000) which can be done via convex optimization. Furthermore, we do not try to match the demonstrator's risk sensitivity, but instead find a robust policy with respect to uncertainty over the demonstrator's reward function, allowing us to optimize policies that are potentially safer than the demonstrations.

One of the concerns of imitation learning, and especially inverse reinforcement learning, is the possibility of learning an incorrect reward function that leads to negative side-effects, for example, a vacuuming robot that learns that it is good to vacuum up dirt, but then goes around making messes for itself to clean up (Russell and Norvig, 2002). To address negative side-effects, most prior work on safe inverse reinforcement learning takes a minmax approach and seeks to optimize a policy with respect to the worst-case reward function (Syed et al., 2008; Hadfield-Menell et al., 2017; Huang et al., 2018); however, treating the world as if it is completely adversarial (e.g., completely avoiding a novel patch of red flooring because it could potentially be lava (Hadfield-Menell et al., 2017)) can lead to overly conservative behaviors. On the other hand, other work on inverse reinforcement learning and imitation learning takes a risk neutral approach and simply seeks to perform well in expectation with respect to uncertainty over the demonstrator's reward function (Ramachandran and Amir, 2007; Ziebart et al., 2008). This can result in behaviors that are overly optimistic in the face of uncertainty and can lead to policies with high variance in performance which is undesirable in high-risk domains such as medicine or autonomous driving. Instead of assuming either a purely adversarial environment or a risk-neutral one, we propose the first inverse reinforcement learning algorithm capable of appropriately balancing caution with expected performance in a way that reflects the risk-sensitivity of the particular application.

This chapter is outlined as follows. First, we propose Bayesian Robust Optimization for Imitation Learning (BROIL), the first imitation learning framework to directly optimize a policy that balances the expected return and the conditional value at risk under an un-

certain reward function. Second, we propose and compare two instantiations of BROIL: optimizing a purely robust policy with respect to uncertainty and optimizing a policy that minimizes baseline regret with respect to expert demonstrations. Finally, we consider two case-studies that demonstrate that BROIL can achieve better expected return and robustness than existing risk-sensitive and risk-neutral IRL algorithms, as well as providing a richer class of solutions that correctly balance performance and risk based on different levels of risk aversion.

7.1 Preliminaries

This chapter uses different notation from the rest of the thesis which we introduce briefly below. We use uppercase and lowercase boldface characters to denote matrices and vectors respectively.

7.1.1 Markov Decision Processes

We model the environment as a Markov Decision Process (MDP) (Puterman, 2014). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, r, P, \gamma, p_0)$, where $\mathcal{S} = \{1, \ldots, S\}$ are the states, $\mathcal{A} = \{1, \ldots, A\}$ are the actions, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition dynamics, $\gamma \in [0, 1)$ is the discount factor, and $p_0 \in \Delta^S$ is the initial state distribution with Δ^k denoting the probability simplex in k-dimensions.

A policy is denoted by $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. When learning from demonstrations we denote the expert's policy by $\pi_E : \mathcal{S} \to \mathcal{A}$. The rewards received by a policy at each state are $\boldsymbol{r}_\pi$ where $\boldsymbol{r}_\pi(s) = \mathbb{E}_{a\sim\pi(s)}[R(s, a)]$ and the transition probabilities for a policy π: $\boldsymbol{P}_\pi$, treated as a matrix, are defined as: $\boldsymbol{P}_\pi(s, s') = \mathbb{E}_{a\sim\pi(s)}[P(s, a, s')] = \sum_a \pi(a|s)P(s, a, s')$. We denote the state-action occupancies of policy π as $\boldsymbol{u}_\pi \in \mathbb{R}^{S\cdot A}$, where $\boldsymbol{u}_\pi = (\boldsymbol{u}_\pi^{a_1\mathsf{T}}, \ldots, \boldsymbol{u}_\pi^{a_A\mathsf{T}})^\mathsf{T}$ and $\boldsymbol{u}_\pi^a(s) = \mathbb{E}[\sum_{t=0}^\infty \gamma^t \mathbf{1}_{(s_t=s\wedge a_t=a)}]$. If we denote the reward function as a vector $\boldsymbol{r} \in \mathbb{R}^{S\cdot A}$, then expected return of policy π under reward function r is denoted by $\rho(\pi, r) = \boldsymbol{u}_\pi^\mathsf{T}\boldsymbol{r}$.

7.1.2 Linear Reward Functions

We assume, without loss of generality, that the reward function can be approximated as a linear combination of k features $\boldsymbol{r} = \boldsymbol{\Phi}\boldsymbol{w}$, where $\boldsymbol{\Phi} \in \mathbb{R}^{S \cdot A \times k}$ is the linear feature matrix with rows as states and columns as features and $\boldsymbol{w} \in \mathbb{R}^k$. This is a strict generalization of the standard MDP formalism since if $\boldsymbol{\Phi}$ is the identity matrix, then each state-action pair is allowed a unique reward. However, it is often the case that the rewards at different states are correlated via observable features which can be encoded in $\boldsymbol{\Phi}$. The features themselves may be highly nonlinear functions of the states and actions that may have been learned via an auxiliary task as discussed in Chapter 6. Given $\boldsymbol{r} = \boldsymbol{\Phi}\boldsymbol{w}$, we denote the expected discounted feature counts of a policy as $\boldsymbol{\mu}_\pi = \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{u}_\pi$, where $\boldsymbol{\mu}_\pi \in \mathbb{R}^k$. In this case, the return of a policy is given by $\rho(\pi, r) = \boldsymbol{u}_\pi^\mathsf{T}\boldsymbol{\Phi}\boldsymbol{w} = \boldsymbol{\mu}_\pi^\mathsf{T}\boldsymbol{w}$.

7.1.3 Distributions over Reward Functions

We are interested in problems where there is uncertainty over the true reward function. We will model this uncertainty as a distribution over R, the random variable representing the true reward function. This distribution could be a prior distribution $\mathbb{P}(R)$ that the agent has learned from previous tasks (Xu et al., 2019). Alternatively the distribution could be the posterior distribution $\mathbb{P}(R \mid D)$ learned via Bayesian inverse reinforcement learning (Ramachandran and Amir, 2007) given demonstrations D or the posterior distribution $\mathbb{P}(R \mid R')$ learned via inverse reward design given a human-specified proxy reward function R' (Hadfield-Menell et al., 2017). While the distribution over R may have an analytic form, this distribution is typically only available via sampling techniques such as Markov chain Monte Carlo (MCMC) sampling (Ramachandran and Amir, 2007).

7.1.4 Risk Metrics

Value at Risk When dealing with measures of risk we will assume that lower values are worse. Thus, as depicted in Figure 7.1, we will want to maximize the value at risk

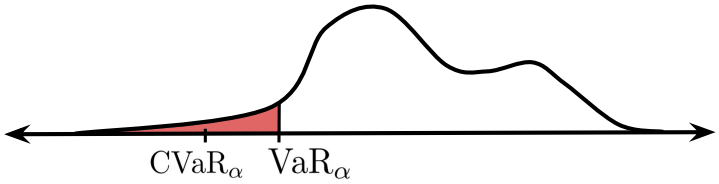


Figure 7.1: VaR_α measures the $(1-\alpha)$-quantile worst-case outcome in a distribution. CVaR_α measures the expectation given that we only consider values less than the VaR_α.

(VaR) or conditional value at risk (CVaR). Given a risk-aversion parameter $\alpha \in [0, 1]$, the VaR_α is the $(1-\alpha)$-quantile worst-case outcome. Thus, VaR_α can be written as $\mathrm{VaR}_\alpha[X] = \sup\{x : \mathbb{P}(X \geq x) \geq \alpha\}$. Typical values of α for risk-sensitive applications are $\alpha \in [0.9, 1]$. Despite the popularity of VaR, optimizing a policy for VaR has several problems: (1) VaR is not convex and results in an NP hard optimization problem (Delage and Mannor, 2010), (2) VaR ignores risk in the tail that occurs with probability less than $(1-\alpha)$ which is problematic for domains where there are rare but catastrophic outcomes, and (3) VaR is not a coherent risk measure (Artzner et al., 1999).

Conditional Value at Risk CVaR is a coherent risk measure (Delbaen, 2002) that is also commonly referred to as average value at risk, expected tail risk, or expected shortfall. For continuous distributions, the CVaR is defined as

$$\mathrm{CVaR}_\alpha[X] = \mathbb{E}\left[X \mid X \leq \mathrm{VaR}_\alpha[X]\right]. \tag{7.1}$$

In addition to being coherent, CVaR is convex, and is a lower bound on VaR. CVaR is often preferable over VaR because it does not ignore the tail of the distribution and it is convex (Rockafellar and Uryasev, 2000).

7.2 Balancing Risk and Return for Safe Imitation Learning

Let Π be the set of all randomized policies, and let $\mathcal{R}$ be the set of all reward functions. Given some function $\psi : \Pi \times \mathcal{R} \to \mathbb{R}$ representing any performance metric for a policy under the unknown reward function $R \sim \mathbb{P}(R)$, we seek to find the policy that is the solution to the following problem:

$$\max_{\pi} \text{CVaR}_{\alpha}[\psi(\pi, R)] \tag{7.2}$$

We now discuss how to solve for the policy that optimizes Equation (7.2). We make use of the one-to-one correspondence between randomized policies $\pi : \mathcal{S} \to \Delta^A$ (where A is the number of actions) and the state-action occupancy frequencies $\boldsymbol{u}_\pi$ (Puterman, 2014). Therefore $\max_\pi \rho(\pi, r)$ corresponds to the following linear program (Puterman, 2014; Syed et al., 2008):

$$\max_{\boldsymbol{u} \in \mathbb{R}^{SA}} \left\{ \boldsymbol{r}^\mathsf{T} \boldsymbol{u} \;\middle|\; \sum_{a \in \mathcal{A}} (\boldsymbol{I} - \gamma \cdot \boldsymbol{P}_a^\mathsf{T}) \boldsymbol{u}^a = \boldsymbol{p}_0, \boldsymbol{u} \geq \boldsymbol{0} \right\} . \tag{7.3}$$

We denote the posterior distribution over samples from $\mathbb{P}(R|D)$ as the vector $\boldsymbol{p}_R$, where each element of $\boldsymbol{p}_R$ represents the probability mass of one of the samples from the posterior distribution, e.g., $\boldsymbol{p}_R[i] = 1/N$ for N sampled reward functions $R_1, R_2, R_3, \ldots R_N$ obtained via MCMC (Ramachandran and Amir, 2007; Brown and Niekum, 2018). Because posterior distributions obtained via Bayesian IRL are usually discrete (Ramachandran and Amir, 2007; Sadigh et al., 2017; Hadfield-Menell et al., 2017; Brown and Niekum, 2018), we cannot directly optimize for CVaR using the definition in Equation (7.1) since this definition only works for atomless distributions (i.e. most continuous distributions). Instead, we make use of the following convex definition of CVaR (Rockafellar and Uryasev, 2000) that works for any distribution (discrete or continuous):

$$\text{CVaR}_{\alpha}[X] = \max_{\sigma} \left(\sigma - \frac{1}{1-\alpha} \mathbb{E}[(\sigma - X)_+] \right) , \tag{7.4}$$

where $(x)_+ = \max(0, x)$ and σ roughly corresponds to the VaR_α (Rockafellar and Urya-

sev, 2000).

Writing the convex definition of CVaR in terms of a the probability mass vector $\boldsymbol{p}_R$, results in the following definition of the CVaR of a policy π:

$$\mathrm{CVaR}_\alpha[\psi(\pi,R)] = \max_\sigma \left(\sigma - \frac{1}{1-\alpha}\boldsymbol{p}_R^\mathsf{T}[\sigma\cdot\mathbf{1} - \boldsymbol{\psi}(\pi,R)]_+\right) , \tag{7.5}$$

where $\boldsymbol{\psi}(\pi,R) = \big(\psi(\pi,R_1),\ldots,\psi(\pi,R_N)\big)^\mathsf{T}$, $[\cdot]_+$ denotes the element-wise non-negative part of a vector: $[\boldsymbol{y}]_+ = \max\{\boldsymbol{y},\mathbf{0}\}$.One of the main insights of this chapter is that, using the same approach as the linear program above, we can formulate (7.2) as the following linear program which can be solved in polynomial time:

$$\max_{\boldsymbol{u}\in\mathbb{R}^{AS},\sigma\in\mathbb{R}} \left\{ \sigma - \frac{1}{1-\alpha}\boldsymbol{p}_R^\mathsf{T}\left[\sigma\cdot\mathbf{1} - \boldsymbol{\psi}(\pi,R)\right]_+ \;\middle|\; \sum_{a\in\mathcal{A}}(\boldsymbol{I} - \gamma\cdot\boldsymbol{P}_a^\mathsf{T})\boldsymbol{u}^a = p_0, \boldsymbol{u}\geq\mathbf{0}\right\} . \tag{7.6}$$

Given the state-action occupancies $\boldsymbol{u}$ that maximize the above objective, the optimal policy can be recovered by appropriately normalizing these occupancies (Puterman, 2014). Thus, the optimal risk-averse IRL policy $\pi^\star$ can be constructed from an optimal $\boldsymbol{u}^\star$ solution to (7.6) as: $\pi^\star(s,a) = {\boldsymbol{u}^\star(s,a)}/{\sum_{a'\in\mathcal{A}}\boldsymbol{u}^\star(s,a')}$.

7.2.1 Balancing Robustness and Expected Return

The above formulation finds a policy that has maximum CVaR. While this makes sense for highly risk-sensitive domains such as autonomous driving (Wulfmeier et al., 2017; Sadigh et al., 2017) or medicine (Kalantari et al., 2020; Asoh et al., 2013), in other domains such as a robot vacuuming office carpets, we may also be interested in efficiency and performance, rather than pure risk-aversion. Even in highly risky situations, completely ignoring expected return and optimizing only for low probability events can lead to nonsensical behaviors that are overly cautious, such as an autonomous car deciding to never merge onto a busy

highway.[13].

To tune the risk-sensitivity of the optimized policy, we generalize Equation (7.6) in order to solve for the policy that optimally balances the expected return and epistemic risk over the reward function. We formalize this via the parameter $\lambda \in [0, 1]$. When $\lambda = 0$ we recover the fully robust policy. When $\lambda \in (0, 1)$ we obtain a soft-robustness, and when $\lambda = 1$ we recover the risk-neutral policy that optimizes for the expected reward function under the posterior. We denote this generalized problem as *Bayesian Robust Optimization for Imitation Learning* or BROIL:

$$\begin{aligned} \underset{\boldsymbol{u} \in \mathbb{R}^{SA}, \sigma \in \mathbb{R}}{\text{maximize}} \quad & (1-\lambda) \cdot \left(\sigma - \frac{1}{1-\alpha} \boldsymbol{p}^{\mathsf{T}} \left[\sigma \cdot \mathbf{1} - \boldsymbol{\Psi}(\pi, R) \right]_+ \right) + \lambda \cdot (\boldsymbol{R}\boldsymbol{p})^{\mathsf{T}} \boldsymbol{u} \\ \text{subject to} \quad & \sum_{a \in \mathcal{A}} \left(\boldsymbol{I} - \gamma \cdot \boldsymbol{P}_a^{\mathsf{T}} \right) \boldsymbol{u}^a = \boldsymbol{p}_0, \quad \boldsymbol{u} \geq \mathbf{0} \,, \end{aligned}$$

where $\boldsymbol{R}$ be a matrix $(S \cdot A) \times N$ where each column of $\boldsymbol{R}$ represents one sample of the vector over rewards for each state and action pair, $\boldsymbol{R}\boldsymbol{p}$ is the mean reward under the posterior, and $\lambda \in [0, 1]$ is a hyperparameter determining how much we value expected return over risk minimization.

7.2.2 Measures of Robustness

BROIL provides a general framework for optimizing policies that trade-off risk and return based on the specific choice of random variable $\psi(\pi, R)$, representing the desired measure of the safety or performance of a policy. We next describe two natural choices for defining $\psi(\pi, R)$.

Robust Objective If we seek a policy that is robust over the distribution $\mathbb{P}(R)$, we should optimize CVaR with respect to $\psi(\pi, R) = \rho(\pi, R)$, the return of the policy. Note that

[13] e.g. "Thwarted on the On-ramp: Waymo Driverless Car Doesn't Feel the Urge to Merge", accessed: 05.19.2020.

R is a random variable so $\rho(\pi, R)$ is also a random variable that depends on the posterior distribution over R and on π. In terms of the linear program above we have $\boldsymbol{\psi}(\pi, R) = \boldsymbol{R}^\mathsf{T}\boldsymbol{u}_\pi$.

Robust Baseline Regret Objective If we have a baseline, such as an expert policy or one or more demonstrated trajectories, we may want maximize CVaR with respect to $\psi(\pi, R) = \rho(\pi, R) - \rho(\pi_E, R)$. This form of BROIL seeks to maximize the margin between the performance of the policy and the performance of the demonstrator. Rather than seeking to match the risk of the demonstrator (Lacotte et al., 2019), the Baseline Regret form of BROIL baselines its performance with respect to the random variable $\rho(\pi_E, R)$, while still trying to minimize tail risk. In terms of the linear program above we have $\boldsymbol{\psi}(\pi, R) = \boldsymbol{R}^\mathsf{T}(\boldsymbol{u}_\pi - \boldsymbol{u}_E)$. In practice, we typically only have samples of expert behavior rather than a full policy. In this case, we compute the empirical expected feature counts using a set of demonstrated trajectories $D = \{\tau_1, \ldots, \tau_m\}$ to get $\hat{\boldsymbol{\mu}}_E = \frac{1}{|D|}\sum_{\tau \in D}\sum_{(s_t,a_t)\in\tau}\gamma^t\phi(s_t, a_t)$, where $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward features. We then solve the above linear program with $\boldsymbol{\psi}(\pi, R) = \boldsymbol{R}^\mathsf{T}\boldsymbol{u}_\pi - \boldsymbol{W}^\mathsf{T}\hat{\boldsymbol{\mu}}_E$, where $\boldsymbol{W}$ is a matrix of size k-by-N where each column is a feature weight vector $\boldsymbol{w} \in \mathbb{R}^k$ corresponding to each linear reward function sampled from the posterior.

7.3 Experiments

7.3.1 Zero-shot Robust Policy Optimization

We first consider the case where an agent wants to optimize a robust policy with respect to a prior over reward functions without access to expert demonstrations. This prior could come from historical data or from meta-learning on similar tasks (Xu et al., 2019).

We consider the machine replacement problem, a common problem in the robust MDP literature (Delage and Mannor, 2010). In this problem, there is a factory with a large number of machines with parts that are expensive to replace. There is also a cost

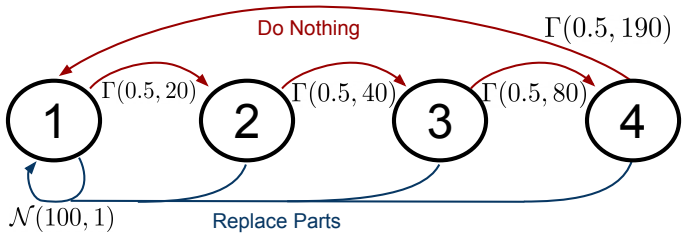
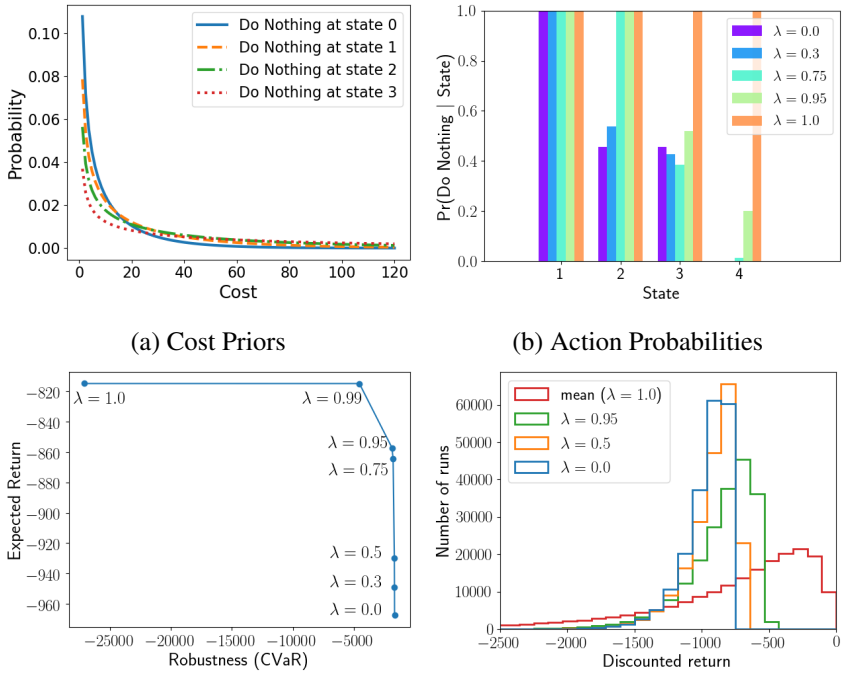


Figure 7.2: Machine Replacement MDP

associated with letting a machine age without replacing parts as this may cause damage to the machine, but this cost is uncertain. We model this problem as the MDP shown in Figure (7.2) with 4 states that represent the normal aging process of the machine, two actions in each state (replace parts or do nothing), discount factor $\gamma = 0.95$, and uniform initial state distribution. The prior distribution over the cost of the Do Nothing action is modeled as a gamma distribution $\Gamma(x, \theta)$, resulting in low expected costs but increasingly large tails as the machine ages. The prior distribution over the cost of replacing a part is modeled using a normal distribution.

Because we have no demonstrations, we use the Robust Objective version of BROIL (Section 7.2.2). We sampled 2000 reward functions from the prior distributions over costs and computed the CVaR optimal policy with $\alpha = 0.99$ for different values of λ. Figure 7.3 (a) shows the action probabilities of the optimal policy under different values of λ, where $\mathbb{P}(\text{Replace Parts}) = 1 - \mathbb{P}(\text{Do Nothing})$. Setting $\lambda = 1$ gives the optimal policy with respect to the mean reward under the reward posterior. This policy is risk-neutral and chooses to never repair the machine since the mean of the gamma distribution is $x \cdot \theta$, so in expectation it is optimal to do nothing. As λ decreases, the optimal policy hedges more against tail risk via a stochastic policy that sometimes repairs the machine. With $\lambda = 0$, we recover the robust optimal policy that only seeks to optimize CVaR. This policy is maximally risk-sensitive and chooses to probabilistically repair the machine in states 2 and 3 and always repair in state 4 to avoid the risk of doing nothing and incurring a possibly high cost. Figure 7.3 (b) shows the efficient frontier of Pareto optimal solutions. BROIL



(a) Cost Priors

(b) Action Probabilities

(c) Efficient Frontier

(d) Return Distributions

Figure 7.3: Risk-sensitive ($\lambda \in [0, 1)$) and risk-neutral ($\lambda = 1$) policies for the machine replacement problem. Varying λ results in a family of solutions that trade-off conditional value at risk and return. The risk-neutral policy has heavy tails, while BROIL produces risk-sensitive policies that trade-off a small decrease in expected return for a large increase in robustness (CVaR).

achieves significant improvements in robustness by sacrificing a small amount of expected utility. Figure 7.3 (c) shows that the BROIL policies with $\lambda < 1$ have much smaller tails than the policy that only optimizes with respect to the expected rewards.

7.3.2 Ambiguous Demonstrations

Next we consider the case where the agent has no prior knowledge about the reward function, but where demonstrations are available. In particular, we are interested in the case where demonstrations cover only part of the state-space, so even after observing a demonstration there is still high uncertainty over the true reward function. To clearly showcase the

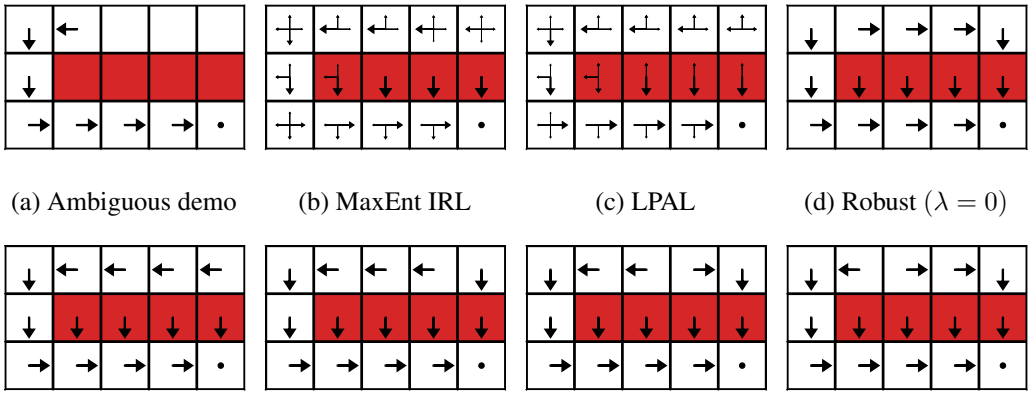
benefits of BROIL, we constructed the MDP shown in Figure 7.4 where there are two features (red and white) with unknown costs, a terminal state in the bottom right, and $\gamma = 0.95$. Actions are in the four cardinal directions with deterministic dynamics. The agent observes the demonstration shown in Figure 7.4 (a) that demonstrates some preference for the white feature over the red feature and a preference for exiting the MDP. However, the demonstration does not provide sufficient information to know what to do in the top right states where demonstrator actions are unavailable. In particular, the agent does not know the true cost of the red cells and whether taking the shortest path from the top right states to the terminal state is optimal. We demonstrate that BROIL results in much more sensible policies across a spectrum of risk-sensitivities, than other state-of-the-art approaches.

Given the single demonstration, we generated 2000 samples from the posterior $P(R|D)$ using Bayesian IRL (Ramachandran and Amir, 2007). We compare against the state-of-the-art risk-sensitive, maxmin algorithm, LPAL, proposed by Syed et al. (Syed et al., 2008) and the state-of-the-art risk-neutral Maximum Entropy IRL algorithm (Ziebart et al., 2008). Shown in Figure 7.4 are the optimal policies for MaxEnt IRL (Ziebart et al., 2008), LPAL (Syed et al., 2008), and BROIL using the robust and baseline regret formulations with $\alpha = 0.95$. We plotted the unique policies and a sample λ that results in each policy. Note that $\lambda = 1$ is equivalent to solving for the optimal policy for the mean reward Figure 7.4 (h). The baseline regret formulation uses the expert feature counts to baseline risk and seeks to completely avoid the red feature for $\lambda = 0$. As λ increases, the baseline regret policy is more willing to take a shortcut to get to the terminal state in the bottom right corner. Conversely, the robust policy takes the shortcut through the far right red cell which balances the risk of the red feature with the knowledge that the white feature is likely to also have high cost.

To better understand the differences between these approaches without committing to a particular ground-truth reward function, we examine each algorithm's performance across the posterior distribution $\mathbb{P}(R|D)$. Figure 7.5 (a) shows $\psi(\pi, R) = \rho(\pi, R)$ sorted

from smallest to largest when evaluated under each sample from the posterior. Figure 7.5 (b) shows the results when $\psi(\pi, R) = \rho(\pi, R) - \rho(\hat{\boldsymbol{\mu}}_E, R)$. LPAL is similar to the baseline regret formulation of BROIL in that it seeks to optimize a policy that performs better than the demonstrator; however, unlike BROIL, LPAL uses a fully adversarial maxmin approach that penalizes the biggest deviation from the demonstrated feature counts (Syed et al., 2008). This results in always avoiding red cells, but also trying to exactly match the feature counts of the demonstration. This feature count matching results in a highly stochastic policy that does not always terminate quickly. MaxEnt IRL is completely risk-neutral, but also seeks to explicitly match feature counts while maintaining maximum entropy over the policy actions. This results in a highly stochastic policy that sometimes takes shortcuts through the red cells, but also sometimes takes actions that move it away from the terminal state.

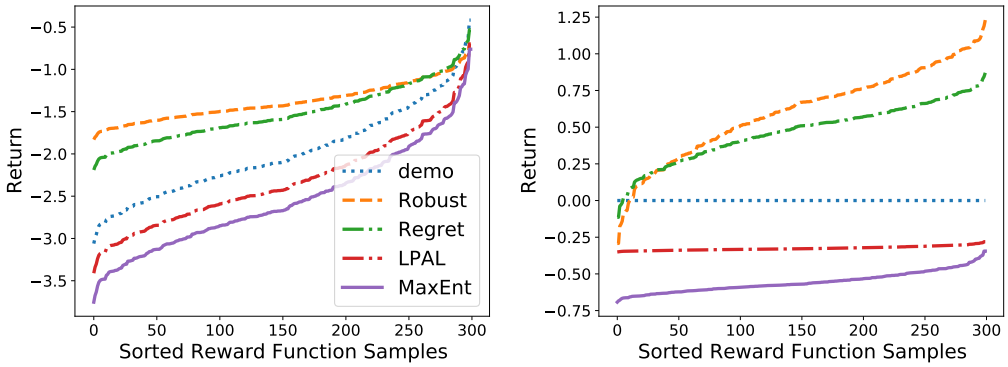
Figure 7.5 shows that both formulations of BROIL significantly outperform MaxEnt IRL and LPAL. The return distribution of the robust BROIL policy is flatter than the other policies as it attempts to find a policy that performs well in the 5% worst-case under all reward functions and needs to be robust to posterior samples that put high costs on white cells and only slightly higher costs on red. On the other hand the Baseline Regret formulation computes risk under the posterior with respect to the expected feature counts $\hat{\boldsymbol{\mu}}_E$ of the demonstrator. This makes reward function hypotheses that would lead to entering red states more risky since the demonstrator only visited white states. The regret formulation seeks to maximize the margin between the return of the baseline regret policy and the return of the demonstration over the posterior. Thus, the regret policy tracks the performance of the baseline more than the robust policy as shown in Figure 7.5(a). As shown in Figure 7.5 (b), the regret formulation has better tail performance with respect to the posterior baseline regret. Figure 7.5 (c) shows the efficient frontier for the baseline regret formulation and shows that BROIL dominates LPAL and MaxEnt IRL with respect to both expected return and robustness.



(a) Ambiguous demo (b) MaxEnt IRL (c) LPAL (d) Robust ($\lambda = 0$)

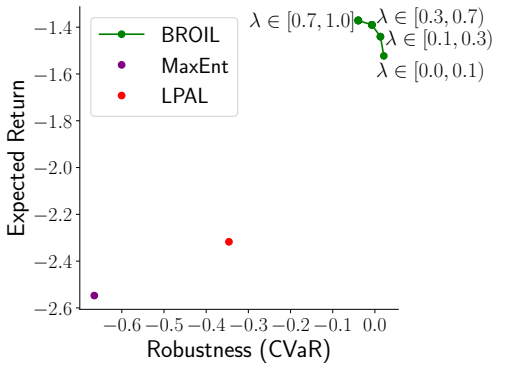
(e) Regret ($\lambda = 0$) (f) Regret ($\lambda = 0.1$) (g) Regret ($\lambda = 0.3$) (h) B-IRL ($\lambda = 1$)

Figure 7.4: When demonstrations BROIL results in a family of solutions that balance return and risk based on the value of λ. (a) Ambiguous demonstration that does not convey enough information to determine how undesireable the red states are. (b-c) MaxEnt IRL (Ziebart et al., 2008) and LPAL (Syed et al., 2008) results in stochastic policies where size of arrow reprents probability. (d) The robust policy with $\lambda = 0$ balances the goodness and badness of red and prefers taking a shortcut. (e-g) The regret policy avoids red for small λ. (h) The optimal policy for the mean reward ($\lambda = 1$) takes a short cut through red cells.



(a) Robustness

(b) Performance over baseline



(c) Efficient frontier

Figure 7.5: Sorted return distributions over the posterior for the BROIL Robust and Baseline Regret policies compared to the return distributions of the demonstration, MaxEnt IRL (Ziebart et al., 2008), LPAL (Syed et al., 2008). The robust policy attempts to maximize worst-case performance over the posterior. The baseline regret also seeks to maximize worst-case performance but relative to the demonstration.

7.4 Summary

In this chapter we proposed Bayesian Robust Optimization for Imitation Learning (BROIL), a method for optimizing a policy to be robust to conditional value at risk under an unknown reward function. Our results show that BROIL can result in better overall performance than existing risk-sensitive maxmin (Syed et al., 2008) and risk-neutral (Ziebart et al., 2008) approaches to IRL. Our approach balances expected return and conditional value at risk over a reward function posterior distribution to produce a family of robust solutions parameterized by the risk-aversion of the user. While our experiments focused on simpler case-studies, where obtaining a posterior via classical Bayesian IRL (Ramachandran and Amir, 2007) is tractable, BROIL allows robust policy optimization given any distribution over reward functions. Thus, our results in Chapter 6 could be used to obtain a posterior distribution over reward functions without requiring an MDP solver, assuming access to high-confidence preferences over demonstrations. We leave this extension as an area for future work.

This chapter focused on the case where there is a fixed posterior distribution over reward functions and an agent wants to optimize their policy with respect to this fixed distribution. However, in more interactive settings, the imitation learning agent may be able to request additional demonstrations or critiques of its behavior. In the next chapter we examine the setting of interactive policy improvement, where an imitation learning agent can actively query for additional help from a demonstrator. In particular, we will propose a novel risk-aware query strategy based on the high-confidence policy evaluation framework described in Chapter 4.

Chapter 8

Risk-Aware Active Inverse Reinforcement Learning

In this chapter we focus on sample efficient safe imitation learning in an interactive setting where a learning agent is able to interact with a human in order to infer the human's intent and optimize a policy that is safe with respect to the learner's epistemic uncertainty over the demonstrator's intent. This chapter presents Contribution 8 of this dissertation: active learning algorithms that generate risk-aware queries for robust policy improvement. Our proposed approach utilizes the high-confidence performance bounds from Chapter 4 to synthesize risk-aware queries that ask for help in states from which the robot thinks it could have high generalization error under the demonstrator's reward function. Previously, in Chapter 6, we briefly highlighted the benefits of introducing risk-informed active queries to refine an imitation learner's posterior distribution over reward functions. In the sections that follow, we develop and explore the idea of risk-informed queries in more detail. While Chapter 7 focuses on the problem of optimizing a policy with regards to a fixed distribution on reward functions, the current chapter focuses on the orthogonal problem of how to select active queries in an risk-aware manner in order to refine an existing distribution over reward

functions. [14]

When a human gives demonstrations to a robot or other autonomous agent, it can be difficult for a human to understand what demonstrations would be most informative to provide to a robot, due to inherent physical, perceptual, and representational differences. To address this issue, *active* IRL algorithms (Lopes et al., 2009; Cohn et al., 2011; Cui and Niekum, 2018) have been proposed that reason about uncertainty and information gain to select queries that are expected to be the most informative under certain criteria. Existing active IRL algorithms aim to minimize uncertainty over policy (Lopes et al., 2009), minimize uncertainty over possible reward functions (Cui and Niekum, 2018), or maximize expected gain in policy value (Cohn et al., 2011). To allow robots to be programmed by non-experts, it is crucial that robots can reason about risk and generalization error given limited, ambiguous demonstrations. However, previous active IRL algorithms do not consider the risk of the actual policy learned by the robot. As an example of a demonstration that could lead to high generalization error, consider a human giving a single demonstration placing a vase on the center of a table. There are many plausible motivations for this demonstration: the vase should be in the center of table, the vase can be anywhere on the table, the vase should be above a cup, etc. In this chapter we seek to addresses the following question: ***How can a robot that learns from demonstrations actively query for help in states where its learned policy has the potential for high generalization error under the demonstrator's true reward function?***

8.1 Methodology

We propose a general framework for risk-aware active queries based on the Value-at-Risk policy loss bounds for LfD proposed in Chapter 4. Our approach generates active queries that seek to minimize the policy loss Value-at-Risk of the robot's learned policy.

[14] This chapter contains work that was done in collaboration with Yuchen Cui and Scott Niekum and was previously published at CoRL 2018 (Brown et al., 2018).

8.1.1 Bounding the Performance of a Policy Given Demonstrations

As mentioned in the related work, existing active IRL approaches typically do not explicitly use performance as a query strategy. By applying and extending the results from Chapter 4, we demonstrate that it is possible to use performance-based active learning to improve a policy learned purely from demonstration.

In Chapter 4 we used samples from $P(R|D)$ to compute the policy loss α-*Value at Risk* (α-quantile worst-case outcome) (Jorion, 1997). Given an MDP\R, a policy to evaluate π_{eval}, and a set of demonstrations D, the policy loss α-VaR provides a high-confidence upper bound on the α-worst-case policy loss incurred by using π_{eval} instead of π^*, where π^* is the optimal policy under the demonstrator's true reward function R. The policy loss of executing π_{eval} under the reward R is given by the Expected Value Difference:

$$\text{EVD}(\pi_{eval}, R) = V_R^* - V_R^{\pi_{eval}}. \tag{8.1}$$

However, IRL is ill-posed—there are an infinite number of reward functions that explain any optimal policy (Ng and Russell, 2000). Thus, any method that attempts to bound the performance of a policy given only demonstrations should account for the fact that there is never one "true" reward function, but rather a set of reward functions that motivate a demonstration. To bound the policy loss α-VaR of an evaluation policy π_{eval} given only demonstrations, we proposed in Chapter 4 to use Bayesian IRL to generate samples of likely reward functions, R, from the posterior $P(R|D)$. Each sample reward function produces a sample policy loss, $\text{EVD}(\pi_{eval}, R)$, and these policy losses can then be sorted to obtain a single sided $(1-\delta)$-confidence bound on the policy loss α-VaR. This allows us to estimate of the Value-at-Risk over any distribution $P(R|D)$.

8.1.2 Risk-Aware Active Queries

The method presented in Chapter 4 and summarized above, works for any evaluation policy π_{eval}. In an active learning setting, we argue that the most useful policy to evaluate is the robot's best guess of the optimal policy under the demonstrator's reward. Thus, we use $\pi_{\text{eval}} = \pi_{\text{MAP}}$ where π_{MAP} is the optimal policy corresponding to R_{MAP}, the maximum a posteriori reward given the demonstrations so far; however, our general approach can easily be applied to any policy learned from demonstrations.

Rather than directly using the approach described in Chapter 4 and Section 8.1.1 to bound the expected α-Value-at-Risk policy loss of the policy π_{MAP}, we instead generate risk-aware active queries by calculating the α-VaR for each potential query state s and select the state with the highest policy loss α-VaR as the query. The policy loss of a state s under π_{MAP} given reward R is:

$$Z = \text{EVD}(s, R \mid \pi_{\text{MAP}}) = V_R^*(s) - V_R^{\pi_{\text{MAP}}}(s) \tag{8.2}$$

where π^* denotes the optimal policy under R. Therefore, the α-VaR for a state can be found using an extension of the method discussed in Section 8.1.1. We first sample rewards $R \sim P(R|D)$ using Bayesian IRL (Ramachandran and Amir, 2007), then for each sampled R and each potential query state s, we compute a sample policy loss Z using Equation 8.2. These samples for each state can then be used to obtain one-sided confidence bounds on the α-VaR for each state as discussed in Chapter 4. Note that the state value functions $V_R^*(s)$ can be directly computed from the optimal Q-value functions obtained during MCMC sampling during Bayesian IRL.

We consider two types of interactions with the demonstrator: active action queries and active critique queries. An active action query is where the robot proposes a state as its query and a human demonstrator is expected to provide the correct action to take at that state. The active learning process is summarized in Algorithm 5. The objective of our

Algorithm 5 Action Query ActiveVaR(Input: MDP\R, D, α, ϵ; Output: R_{MAP}, π_{MAP})

1. Sample a set of reward functions R by running Bayeisan IRL with input D and MDP\R;
2. Extract the MAP estimate R_{MAP} and compute π_{MAP};
3. **while** *true*:
   (a) $s_k = \arg\max_{s_i \in S}(\alpha\text{-VaR}(s_i, \pi_{MAP}))$;
   (b) Ask for expert demonstration a_k at s_k and add (s_k, a_k) into demonstration set D;
   (c) Sample a new set of rewards R by running Bayesian IRL with updated D;
   (d) Extract the MAP estimate R_{MAP} and compute π_{MAP};
   (e) **break** if $\max_{s_i \in S}(\alpha\text{-VaR}(s_i, \pi_{MAP})) < \epsilon$;
4. **return** R_{MAP}, π_{MAP}

active learning algorithm is to minimize the worst-case generalization error of the learned policy, with as few queries as possible. To do this we compute the α-VaR for each starting state state (sampling states for continuous environments). The state with the highest α-VaR under $P(R \mid D)$ is the state where the robot's policy has the highest α-quantile worst-case policy loss. Thus, selecting this state as the active query is a good approximation for our objective.

In the second type of query, the robot demonstrates a trajectory to the user and asks them to critique the demonstration by segmenting the demonstration into desirable and undesirable segments as proposed by Cui and Niekum (2018). To generate a trajectory for critique, the state with the highest α-VaR policy loss is computed. A trajectory is then generated by executing the MAP policy starting at the selected state. This active learning process is summarized in Algorithm 6, where the demonstrator provides a critique of one of the agent's proposed trajectories in the form of segmenting the trajectory into optimal and suboptimal segements.

Both of the above active learning algorithms repeat until the α-VaR of the robot's

Algorithm 6 Critique Query ActiveVaR(Input: MDP\R, D, α, ϵ; Output: R_{MAP}, π_{MAP})

1. Sample a set of reward functions R by running Bayeisan IRL with input D and MDP\R;
2. Extract the MAP estimate R_{MAP} and compute π_{MAP};
3. **while** *true*:
   (a) $s_k = \arg\max_{s_i \in S}(\alpha\text{-VaR}(s_i, \pi_{MAP}))$;
   (b) Ask for critique of trajectory from π_{MAP} of length L that starts at state s_k;
   (c) Update D with positive and negative segments from the critique;
   (d) Sample a new set of rewards R by running Bayesian IRL with updated dataset D of positive and negative examples (Cui and Niekum, 2018);
   (e) Extract the MAP estimate R_{MAP} and compute π_{MAP};
   (f) **break** if $\max_{s_i \in S}(\alpha\text{-VaR}(s_i, \pi_{MAP})) < \epsilon$;
4. **return** R_{MAP}, π_{MAP}

learned policy falls below the desired safety threshold ϵ. We call our framework Risk-Aware Active IRL and refer to the corresponding active learning algorithms as *ActiveVaR* in future discussion.[15]

8.1.3 Example

Given samples of reward functions from running Bayesian IRL, there are many different statistical measurements that can serve as the objective function for active learning. Methods such as those proposed by Lopes et al. (2009) and Cui and Niekum (2018) are purely exploratory since they reason only about statistical measures of uncertainty over the reward function posterior instead of the performance of a specific policy when selecting active queries. In contrast, our proposed framework instead uses focused exploration based on high-confidence bounds on the policy loss of the agent's policy.

[15] Our implementation of ActiveVaR can be found at: https://github.com/Pearl-UTexas/ActiveVaR.git

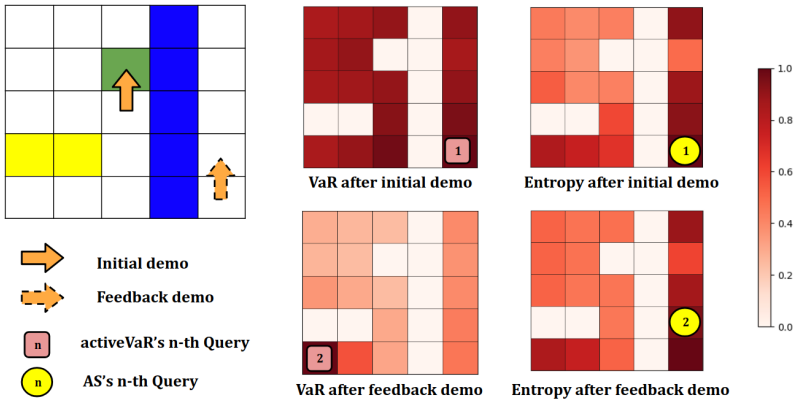


Figure 8.1: Comparison of active action queries based on performance loss risk or action entropy. The example gridworld has four different unknown features denoted by the yellow, green, white, and blue colors of the cells. White states are legal initial states. AS is the active learning algorithm proposed by Lopes et al. (Lopes et al., 2009) and activeVaR is our proposed active action query algorithm. The first two active queries proposed by each algorithm are annotated on the heatmaps of VaR and entropy values after each iteration. For heatmaps, all values are normalized from 0 to 1.

The example in Figure 8.1 shows how focusing on Value-at-Risk, rather than minimizing uncertainty over actions as in the Active Sampling (AS) algorithm (Lopes et al., 2009), leads to more intuitively intelligent queries. In this example, there are four indicator features denoted by the white, yellow, green, and blue colors on the cells. Given one initial demonstration going into the green state from a white state, AS and ActiveVaR both pick the bottom right white state as their first query. However, for the second query, AS picks another state next to a blue feature while ActiveVaR picks the state next to a yellow feature. Active queries based on VaR allow the IRL agent to understand that blue feature is unavoidable from all the rightmost states so there is no point asking for more demonstrations from those states while AS only reasons about action entropy.

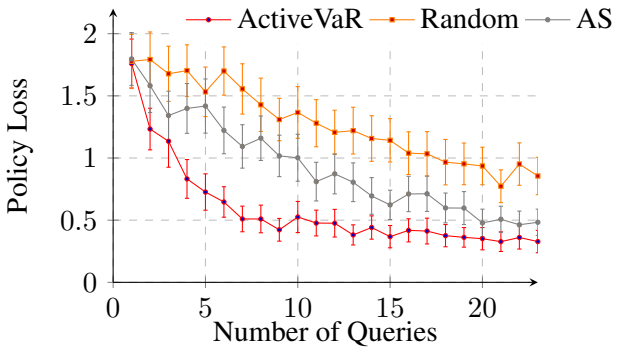


Figure 8.2: A comparison of different active action query strategies. ActiveVar (ours) outperforms action queries chosen at random as well as action queries chosen based on action entropy (AS) as proposed by Lopes et al. (2009). Results show averaged policy losses in 8×8 gridworlds with 48 features.

8.2 Experiments

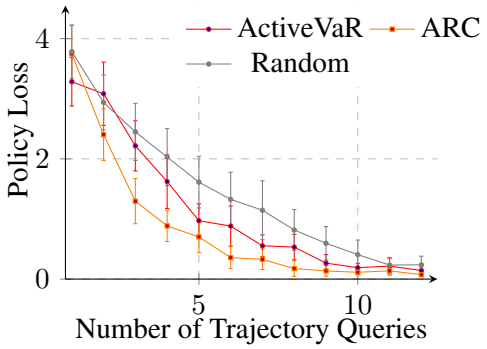
8.2.1 Gridworld Active Action Queries

We first evaluate ActiveVaR on active action queries in which the active learning algorithm selects a state and asks the demonstrator for an action label. The Active Sampling (AS) algorithm proposed by Lopes et al. (2009) can only work with action queries. AS is not as computationally efficient as activeVaR, but is much more computationally efficient than methods that require sampling hypothesis probability distributions (Cui and Niekum, 2018; Cohn et al., 2011). We conducted experiments in simulated gridworlds of size 8x8 with 48 random continuous features. The ground-truth feature weights are generated randomly and normalized such that the L1 norm of the weight vector is 1. The expected losses in return comparing to the optimal policy are measured and plotted in Figure 8.2, where *Random* is a baseline that selects a random state as an active query per iteration. ActiveVaR is able to rapidly reduce the policy loss over iterations since it directly optimizes for performance.

8.2.2 Gridworld Active Critique Queries

The previous section demonstrated that action queries using a risk-aware active learning approach based on expected performance loss, or risk, outperforms standard entropy-based queries. We now demonstrate that our risk-aware active learning framework also allows active critique queries. Cui and Niekum (2018) proposed a novel active learning algorithm, ARC, where the robot actively chooses sample trajectories to show the human and the human critiques the trajectories by segmenting them into good and bad segments. This type of active query can be more natural for a human than giving an out of context action for a state and only requires the human to be able to recognize, not demonstrate, desirable and undesirable behavior. However, synthesizing trajectories for critique requires costly Bayesian inference over possible segmentations. Additionally, it is purely exploratory since it reasons only about information gain in the reward belief space without reasoning about the performance of current learned policy.

To generate risk-aware performance-based trajectories we examine the robot's policy and evaluate per-state policy loss 0.95-VaR. Because the VaR is calculated based on the value of executing the robot's current policy from that state, we sample a trajectory from the robot's policy starting at that state and ask the demonstrator to critique it. We conducted experiments in 8x8 gridworlds with 48 different continuous features and allow each algorithm to generate a trajectory query of length 8 per critiquing iteration. As baselines, *Random* selects an active query by rolling out the current MAP policy from a randomly selected state. As shown in Figure 8.3, while ActiveVaR's performance is not as good as that of ARC per number of trajectory queries, the time required to run ActiveVaR is much less than that of ARC. Because ARC's inference algorithm is sequential, not parallelizable, ActiveVaR is more practical for real-time active learning scenarios, while also outperforming random queries. Additional experiments were performed to highlight cases when ActiveVaR outperforms Random by a much larger margin (see Appendix G.1).



(a) Averaged policy losses

Algorithm	Avg. Time (s)
Random	0.0015
ActiveVaR	0.0101
ARC	865.6993

(b) Timing for one iteration of each algorithm

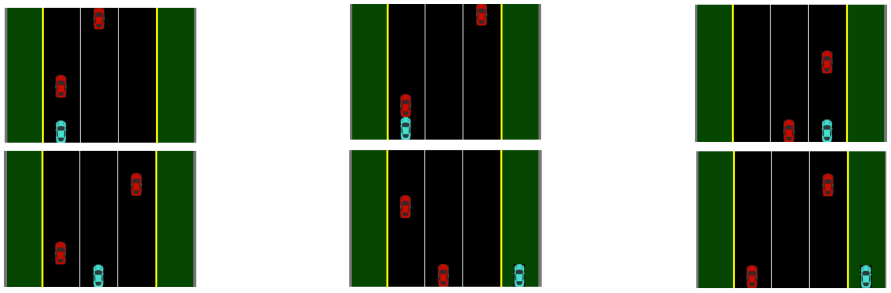
Figure 8.3: Active critique queries in 8×8 gridworlds with 48 features.

8.2.3 Active Imitation Learning for a 2D Highway Driving Task

We also evaluated ActiveVaR for learning from demonstration in a 2D driving simulator, in which the transition dynamics are not deterministic and no ground truth reward function is specified. Human demonstrations are provided by controlling the agent (ourselves) via keyboard. In the 3-lane driving simulator, the controlled agent moves forward two times faster than the two other agents on the road and has three available actions at any given time: moving left, move right, or stay. The states are approximated using discrete features including lane positions and distances to other agents. Figure 8.4 shows the generated active action queries (high-0.95-VaR states) after varying amounts of initial demonstrations are provided. The provided human trajectories demonstrate a safe driving style by avoiding collisions and staying on the road (black lanes). As increasing amounts of initial demonstrations are provided, the active queries start to explore less-frequent states and corner cases that were not addressed in the initial training data.

8.2.4 Robot Table Setting Task

Finally, we consider a robot learning to set a table based on a demonstrator's preferences (see Figure 8.5). We model the reward function as a linear combination of Gaussian radial



(a) 5 Steps of Demos (b) 10 Steps of Demos (c) 20 Steps of Demos

Figure 8.4: Active action queries in a 2D highway driving task after different numbers of initial human demonstrations. Initial states are randomly sampled and evaluated; high risk states are selected as active action queries.

basis functions. Given k items on the table, we assume the reward for placement location x is given by

$$R(x) = \sum_{i=1}^{k} w_i \cdot \text{rbf}(x, c_i, \sigma_i^2) \tag{8.3}$$

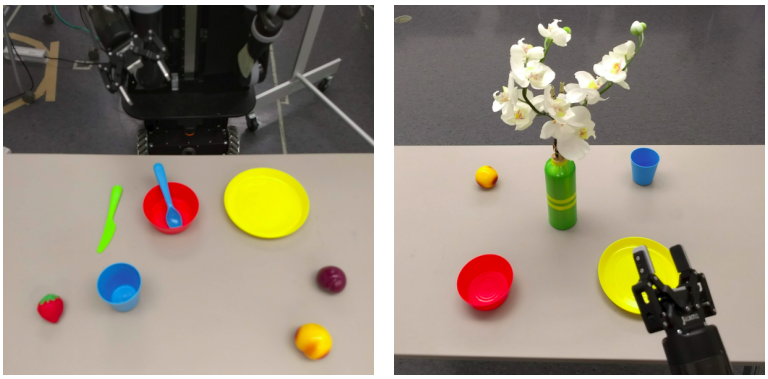
where $\text{rbf}(x, c, \sigma^2) = \exp(-\|x - c\|^2/\sigma^2)$.

The demonstrator first gives a demonstration from an initial configuration C_0. The robot then needs to infer the correct reward function that matches the demonstrator's intention. We consider an active approach where the robot can generate a novel configuration C_i and ask the demonstrator where it should place the item. The robot hypothesizes multiple configurations C_i and picks the configuration C^* that has maximum 0.95-VaR over its current best guess of the demonstrator's policy.

Given an RBF reward function, the robot needs to estimate an optimal placement position. To calculate the optimal position we use gradient ascent with random restarts and pick the best position. The gradient of the reward with respect to the position x is given by:

$$\nabla_x R(x) = \sum_{i=1}^{k} w_i \cdot \text{rbf}(x, c_i, \sigma_i^2) \left(\frac{-2x + 2c_i}{\sigma_i^2} \right). \tag{8.4}$$

MCMC is used to estimate the posterior $P(R|D)$ and determine the MAP reward



(a) Place Spoon (b) Place Vase

Figure 8.5: Setting the table task. (a) Robot actively requests demonstration learning preferences for (a) placing a spoon in the bowl and (b) placing the flower vase in the center of the table.

R_{MAP}, which is used as the best guess of the demonstrator's intent. Given the MCMC estimate of the reward posterior, the robot samples random table configurations C_j. For each random configuration, the robot computes the α-VaR by first finding the best placement position x^*_{MAP} given by the MAP reward function, and then evaluating this placement position over the estimated posterior found using MCMC. This is done by calculating the placement loss, $\text{Loss}_i = \|x^i_{R_i} - x^*_{\text{MAP}}\|_2, \;\; \forall \; R_i \in P(R|D)$, and then sorting to estimate the α-VaR of that configuration. The robot then picks as the query configuration, $C_{\text{query}} = \arg\max_j \alpha\text{-VaR}(C_j)$, and actively asks for a demonstration in this configuration. Note that in this task the epsilon stopping condition can be defined in terms of placement error (distance between where demonstrator would place object and where robot would place object).

Figure 8.6 shows the results of two table arrangement experiments. In the first experiment the demonstrator teaches the robot to place a vase of flowers on a table with 4 existing objects. The preference is to place the vase in the center of the table, while avoiding placing it on top of other objects. In the second experiment, the demonstrator teaches the robot to place a spoon in a bowl on a table with 6 distractor objects. To allow for expressive reward function hypotheses, we model the reward using RBFs centered on all objects

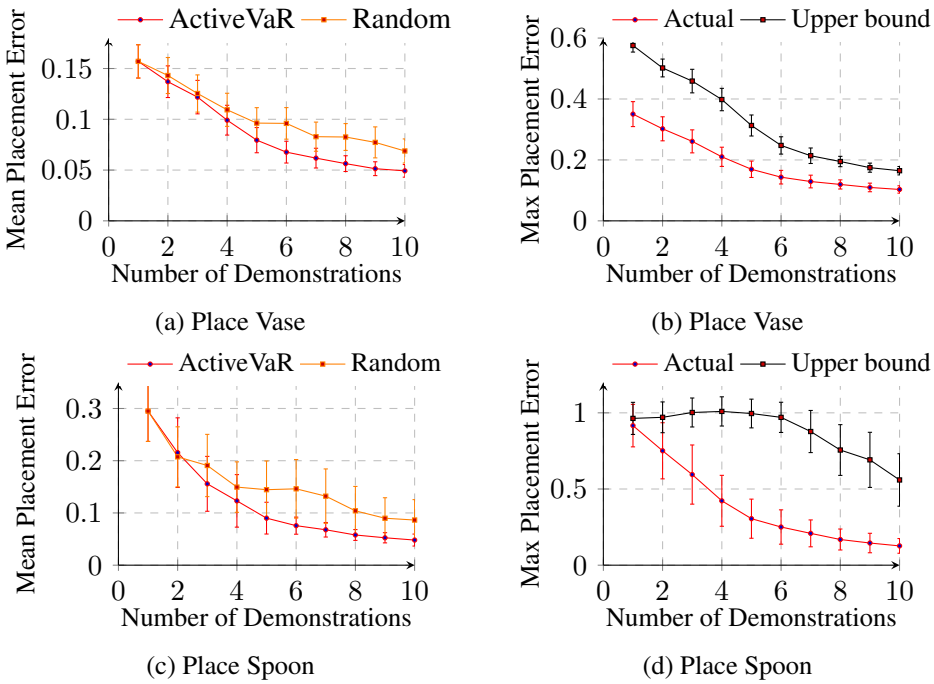
on the table along with 9 fixed RBFs evenly spaced on the table to allow for the possibility of an absolute placement preference. Possible query configurations are randomly generated by changing the position of one of the objects on the table. We generated synthetic ground truth rewards that corresponded to each placement preference and generated synthetic demonstrations using these ground truth rewards. This allows us to rigorously test the active learning process without needing to physically move the objects for each query configuration. We then validated the learned reward function weights on the real robot in a variety of test configurations.

Figures 8.6(a) and (c) show a comparison of random queries with actively querying the configuration with maximum 0.95-VaR out of 50 randomly generated configurations. Choosing risk-aware queries over random queries results in smaller generalization error. To test whether 0.95-VaR provides a meaningful and accurate upper bound on the true performance of a policy we compared the actual worst-case placement loss under for the robot's current policy after each demonstration with the 0.95-VaR upper bound. Figures 8.6(b) and (d) demonstrate that the 0.95-VaR upper bound accurately upper bounds the actual worst-case performance and that this bound becomes tighter as more demonstrations are received.

8.3 Choosing an Intuitive Stopping Condition

The ϵ stopping criterion in our work is based on an upper bound on performance loss. We know of no other work that has proposed or used such a stopping condition. Without context, an ϵ stopping condition based on raw policy loss may not be easy to select; however, policy loss can be normalized to obtain a percentage that is more semantically meaningful. The normalization can be computed as

$$\text{normalized EVD}(s, R \mid \pi_{\text{MAP}}) = \frac{V_R^{\pi^*}(s) - V_R^{\pi_{\text{MAP}}}(s)}{V_R^{\pi^*}(s)}. \tag{8.5}$$



(a) Place Vase

(b) Place Vase

(c) Place Spoon

(d) Place Spoon

Figure 8.6: Results for learning to place a flower vase and learning to place a spoon on a cluttered table. Active queries in (a) and (c) result in lower error than random queries. The 0.95-VaR placement error bound shown in (b) and (d) provides an upper bound on the actual maximum placement error. All results are averaged from 100 trials with 0.5 standard deviation error bars. Placement error is calculated using 200 random test configurations.

Using the normalized EVD in place of EVD in the above algorithms allows us to calculate an upper bound on the normalized Value-at-Risk.

For example, if the normalized α-VaR is 5% then we can say with high-confidence that the expected return of the learned policy is within 5% of the expected return of the optimal policy under the demonstrators reward. This allows epsilon to be set as a fixed percentage such as 5% of 1% depending on risk aversion.

8.4 Summary

In this chapter we proposed the first active IRL technique that is based on the risk-aware queries with respect to the performance of the policy learned from demonstrations. In particular, we used the high-confidence bounds policy loss proposed in Chapter 4 to allow an imitation learning agent to synthesize queries that target areas of the state space where the agent has highest policy loss value at risk. We compared our approach against existing active IRL algorithms and found that our risk-aware approach outperforms entropy based queries in terms of sample complexity and is comparable to active queries based on information gain while requiring three orders of magnitude less computation on the domains we tested. Experiments in simulated 2-d navigation and highway driving domains, as well as robot table placement tasks, demonstrate that risk-aware active queries allow robots to ask for help in areas of the state-space where the robot has the potential for high generalization error. Our approach allows the robot to upper bound its own policy loss and can be used to let a robot to know when it has received enough demonstrations to safely perform a task.

Chapter 9

Future Work

In this chapter we discuss open questions and areas of future work related to the individual research chapters in this dissertation.

9.1 High-Confidence Policy Evaluation for Imitation Learning

In Chapter 4 we formulated high-confidence bounds on the policy loss value at risk as our risk metric; however, this is not the only measure of performance that can be used in our approach. Because our method estimates the posterior distribution over reward functions, any risk measure or loss that is a function of a reward function and a policy can be inserted into our framework in place of EVD. We used value at risk (VaR) because it is and widely used, easy to implement using Monte-Carlo samples, and is a probabilistic analogue to the worst-case feature count baseline described in Section 4.3. However, our proposed methodology can be extended to use other risk measure that can be computed from samples of a distribution. Alternative risk measures such as conditional value at risk (Rockafellar and Uryasev, 2000), entropic risk measure (Föllmer and Knispel, 2011), or semideviations (Ogryczak and Ruszczyński, 1999) could replace VaR in our framework.

Because our high-confidence bounds are based on Bayesian IRL, our method is de-

signed to work with partial demonstrations and allows insertion of domain knowledge as a prior over reward functions. Choi and Kim (2011) showed that many standard IRL algorithms can be transformed into an equivalent Bayesian IRL algorithm by selecting the appropriate likelihood and prior. Thus, our proposed performance bound can be easily extended to use alternative likelihoods and priors that match different assumptions and preferences found in the IRL literature. One interesting avenue of future work is to leverage the work by Zheng et al. (2014) who formulate a Bayesian IRL algorithm that is robust to sparse demonstration noise. This method could be used to obtain the posterior distributions we use for high-confidence bounds, enabling us to formulate more accurate bounds when some of the demonstrations are extremeley noisy or non-sensical. Another interesting area of future work is to better explore the impact of having a prior distribution over reward functions. We assumed flat priors in all of our experiments. Future work should investigate how strong priors affect our bounds. Our results in Hadfield-Menell et al. (2017) propose a Bayesian IRL method based on maximum entropy IRL (Ziebart et al., 2008) that can learn a distribution over rewards without access to demonstrator actions, but cannot learn from partial trajectories. Chapter 4 assumed access to the demonstrator's actions, future work should investigate whether similar results can be obtained when learning only from state observations by formulating a version of Bayesian IRL that can learn from partial state sequences or by learning an inverse dynamics model to recover likely demonstrator actions given state transitions (Torabi et al., 2018a).

Our results in Chapter 4 demonstrated that our high-confidence bounds rely on an appropriate range for the Boltzman rationality parameter β when running Bayesian IRL. This parameter represents the confidence that the learning agent has in the demonstrations being optimal and can have a significant impact on the resulting posterior distribution over reward functions. Zheng et al. (2014) use an Expectation Maximization approach to learn this parameter. Bobu et al. (2018) propose to perform Bayesian inference over the Boltzman parameter β as well as the reward function parameters. Future work should investigate

whether similar approaches can be applied to the setting of high-confidence policy evaluation for imitation learning to learn an appropriate value for β.

9.2 Computationally Efficient Reward Learning from Suboptimal Demonstrations

In Chapter 5 we introduced the T-REX and D-REX algorithms for efficient reward inference from ranked, suboptimal demonstrations. We used undiscounted returns when computing the loss for training T-REX/D-REX. Future work should examine whether adding a discount factor helps to stabilize or improve reward training and/or reinforcement learning. While the pairwise ranking loss that we use is common, it would also be interesting to consider the impact of different losses such as a hinge loss that would enforce a minimum gap in predicted return between ranked trajectories, but not push this gap to be as large as possible.

Other areas of future work include examining the effects of different data augmentation methods such as subsampling or supersampling ranked trajectories, examining the effects of different network architectures, and trying experiments across a wider range of tasks. We used the default OpenAI Baselines (Dhariwal et al., 2017) implementation of PPO along with the default settings for Atari and MuJoCo. Better tuning of these parameters as well as a more in-depth study of the effects of different environmental settings and different policy optimization algorithms would also be interesting.

While we provided separate results for T-REX when using human demonstrations and when using human rankings, we never fully explored the human-factors of having demonstrators provide ranked demonstrations. Studying how to make ranking demonstrations easy for human demonstrators is an open area of research. Additionally, studying whether watching a human demonstrator learn how to perform a task over time provides sufficient preference information to run the time-ordered version of T-REX would also be interesting.

We focused on the setting of pre-ranked demonstrations, but much research focuses on active preference learning during policy optimization (Christiano et al., 2017; Ibarz et al., 2018; Bıyık et al., 2019). Future work should investigate combining pre-ranked demonstrations with active queries during policy optimization. Extending and incorporating ideas from Chapter 8 to enable risk-aware active queries during policy optimization is also an open area of research. Combining T-REX and D-REX with other forms of human interaction such as critiques (Knox and Stone, 2009; Cui and Niekum, 2018) would likely lead to better reward function inference and faster policy optimization.

Because T-REX is a learning from observation method, it can potentially be applied to settings where the action space of the learner and demonstrator are different. Future work should investigate how robust T-REX is to changes in actions and transition dynamics between the demonstrator and imitator. While our D-REX experiments (Section 5.3) use demonstrator actions to perform behavioral cloning, it is also possible that an imitation learning from observation method (Torabi et al., 2018a) could be used to learn an initial cloned policy without requiring demonstrator actions. This would make D-REX robust to action differences between the demonstrator and learner and allow the learner to potentially learn a more efficient policy if its action space and transition dynamics are better suited to the task.

Currently, the main bottleneck in T-REX and D-REX is policy optimization. One potential way to speed up policy optimization is via model-based reinforcement learning. This would allow fast reward inference (via T-REX) and fast trajectory optimization via a high-fidelity model. It would also be interesting to see if T-REX could be combined with standard reinforcement learning for sparse reward tasks in order to automatically shape and densify rewards to speed up learning even when an agent has access to a ground-truth reward function.

9.3 Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences

The high-confidence safety bounds that we propose in Chapter 6 are only safe with respect to the assumptions that we make: good feature pre-training, rapid MCMC mixing, and accurate preferences over demonstrations. Future work includes using exploratory trajectories for better pre-training of the latent feature embeddings, using recent advances in contrastive unsupervised learning (Chen et al., 2020; Srinivas et al., 2020) for feature pre-training, developing methods to determine when a relevant feature is missing from the learned latent space, and using high-confidence performance bounds to perform safe policy optimization in the imitation learning setting.

9.4 Bayesian Robust Optimization for Imitation Learning

Future work includes taking advantage of recent research on efficient non-linear Bayesian reward learning via Gaussian processes (Biyik et al., 2020) and deep neural networks (Chapter 6). Future work also includes investigating natural extensions of our work to continuous state-spaces via approximate linear programming (Petrik, 2010; Pazis and Parr, 2011; Desai et al., 2012). In Chapter 8 we propose an active learning approach that uses VaR bounds to determine risk-aware queries. Future work also includes investigating whether similar queries can be generated using our CVaR optimization method.

In Chapter 7 we propose two variants of BROIL: a robust and a baseline regret. However, there is another possibility. The high-confidence bounds used in chapters 4 and 8 compute policy loss using the expected value difference:

$$\text{EVD}(\pi_{\text{eval}}, R^*) = V^*_{R^*} - V^{\pi_{\text{eval}}}_{R^*}. \tag{9.1}$$

Future work should examine whether robust policy optimization with respect to EVD pro-

vides different results when compared to the robust and baseline regret versions of BROIL. It would also be interesting to combine BROIL with the inverse reward design method proposed by Hadfield-Menell et al. (2017) in order to learn robust policies that balance return and risk with respect to a distribution over reward functions induced by a hand-designed proxy reward function.

9.5 Risk-Aware Active Inverse Reinforcement Learning

Our results in Chapter 8 use classical Bayesian IRL to provide a reward posterior. Future work includes combining the methods from Chapter 6 with risk-aware active queries to enable robust active IRL that scales to more complex imitation learning domains. Our results in Chapter 8 show that high-confidence bounds on VaR provide tight and accurate bounds on the generalization error of an agents policy at test time. In the future, this could be extended to allow an agent to know when it can tell a human demonstrator that it has received enough demonstrations. Performing human factors studies to test this hypothesis is an interesting area of research.

9.6 Additional Frontiers

Throughout this dissertation we have proposed algorithms that provide high-confidence performance bounds based on a reward function posterior or optimize a robust policy with respect to the agent's uncertainty over reward functions. However, this uncertainty is with respect to the agent's internal beliefs and possibly over its own learned representation of the demonstrator's reward function. Thus, the resulting high-confidence bounds and optimized policies may not always conform to a human's intuition about what safe or robust behavior should look like. Advances in explainable AI systems have the potential to mitigate these issues, but explainable AI remains an open area of research and effectively conveying the rationality behind complex risk-return trade-offs is a non-trivial, but important area for

future work.

Much of this thesis has either assumed access to the correct reward function features or assumed that these features can be learned. However, knowing when the agent's reward function features are inadequate is a challenging problem with very few proposed solutions (Bobu et al., 2018). Furthermore, the question of what an agent should do to remedy its reward function feature representation remains an important but open question.

Chapter 10

Conclusion

Recovering the intent of a demonstrator by learning a reward function allows an imitation learning agent to explain the behavior of the demonstrator and also to optimize its own behavior using reinforcement learning, in order to successfully imitate the demonstrator. Recovering a reward function also provides a way to potentially generalize to novel tasks or embodiments, allows a learner to potentially improve upon the performance of the demonstrator, and provides a way to explicity measure costs and reason about safety. However, despite the advantages of imitation learning via inverse reinforcement learning, current inverse reinforcement learning algorithms often have limited real-world applicability because they (1) do not provide practical assessments of safety, (2) often require large numbers of demonstrations, and (3) have high computational costs. This dissertation made several contributions towards addressing these shortcomings. In particular, this dissertation advanced both the theory and applicability of safe imitation learning by focusing on the following question:

How can an autonomous agent efficiently infer the intent of a demonstrator and provide safety guarantees in the form of high-confidence performance bounds with respect to the demonstrator's intent?

In Chapter 4 we presented a theoretical framework and sample efficient algorithm

for safe imitation learning via high-confidence performance bounds. These bounds are orders of magnitude more accurate and sample efficient than the previous state-of-the-art (Syed and Schapire, 2007). However, despite efficiency in terms of demonstrations, these bounds required repeatedly solving a potentially complex reinforcement learning problem in order to obtain a posterior distribution over reward functions.

In Chapter 5 we addressed this inefficiency by proposing reward inference algorithms that do not require an MDP solver or any reward inference time data collection. Furthermore we provided theory on when better-than-demonstrator performance is possible and provided reward learning methods that are computationally efficient, scale to high-dimensional control tasks, and can learn policies that perform significantly better than a suboptimal demonstrator.

Next, in Chapter 6 we addressed the problem of safe and efficient imitation learning by extending and combining our results from chapters 4 and 5 to provide the first Bayesian reward inference algorithm that scales to high-dimensional visual imitation learning tasks and can provide high-confidence policy evaluation.

Finally, we considered the problem of robust policy optimization. In Chapter 7 we proposed an algorithm that directly optimizes a policy so that it balances expected return and conditional value at risk over a reward function posterior distribution. Our resulting algorithm outperforms state-of-the-art risk-neutral and risk-sensitive imitation learning approaches. Next, we considered the problem of robust policy improvement in an interactive setting where an agent can actively query for help in states that have high policy loss risk. We demonstrated that risk-aware queries outperform existing active IRL approaches and allow an agent to upper bound its generalization error at test time.

Algorithms that balance risk and return are have been common in financial applications for a long time, but are just starting to be applied to AI/ML systems. We believe this is a positive trend as many AI/ML applications have risk and return trade-offs that are not always adequately addressed. The work presented in this dissertation allows autonomous

agents to efficiently learn from demonstrations and to express confidence in the performance of their learned policy, based on limited demonstration data. We also demonstrate that imitation learning agents can optimize their policies to reduce risk while maintaining good performance. We believe that this dissertation provides important foundational results towards the development of autonomous agents that can safely and efficiently learn from human demonstrations in risk-sensitive, real-world environments.

10.1 Contributions

In summary, this dissertation made the following contributions to the imitation learning and inverse reinforcement learning literature:

1. Formalization of safe imitation learning via high-confidence policy evaluation (Chapter 4).

2. A sample efficient algorithm for obtaining high-confidence performance bounds for imitation learning (Chapter 4).

3. Theoretical results for better-than-demonstrator imitation learning and preference-based inverse reinforcement learning (Chapter 5).

4. A computationally efficient algorithm for reward learning from suboptimal, ranked observations that scales to high-dimensional tasks and can outperform the demonstrator (Chapter 5).

5. Computationally efficient algorithms for learning to extrapolate intention from unlabeled suboptimal demonstrations (Chapter 5).

6. A deep Bayesian inverse reinforcement learning algorithm that scales to complex, high-dimensional tasks (Chapter 6).

7. An algorithm for Bayesian robust imitation learning that optimizes a policy to balance risk and expected return given a posterior distribution over reward functions. (Chapter 7).

8. An algorithm for risk-aware policy improvement via active inverse reinforcement learning (Chapter 8).

Appendix A

Supplementary Materials for High-Confidence Performance Bounds for Imitation Learning

A.1 Code

Code to reproduce the experiments in Chapter 4 can be found at https://github.com/dsbrown1331/aaai-2018-code.

A.2 L1-norm MCMC Walk

A.2.1 Uniform sampling from L1-unit ball

We derive an algorithm that correctly samples uniformly form the L1-norm unit ball. Our method is a special case of the result by Barthe et al. (Barthe et al., 2005) as detailed in Weisstein (Weisstein, 2017). The general result states that if we wish to sample an element from the L-p ball in d-dimensional space, then we should pick $X_1, \ldots, X_d$ independently

from the pdf

$$P_p(x) = \frac{\exp(-|x|^p)}{2\Gamma(1+p^{-1})} \tag{A.1}$$

where p is the desired p-norm and Γ is the gamma function. Then we draw Y from an exponential distribution with mean 1 and our resulting sample from the L_p norm ball is

$$\frac{(X_1, \ldots, X_n)}{(Y + \sum_{i=1}^{n} |X_i|^p)^{1/p}} \tag{A.2}$$

We wish to sample from the L1-norm boundary, i.e. where the L1-norm is equal to 1. Thus we have $p = 1$ and $Y = 0$ above. This means that we need to sample d numbers independently from the following pdf

$$P_1(x) = \frac{\exp(-|x|)}{2\Gamma(2)} = \frac{\exp(-|x|)}{2}. \tag{A.3}$$

We can sample from this distribution using the inverse CDF sampling method (c.f. Bishop (Bishop, 2006)). To draw samples from this distribution we must compute the inverse of the indefinite integral

$$z = h(x) = \int_{-\infty}^{x} \frac{\exp(-|\hat{x}|)}{2} d\hat{x} \tag{A.4}$$

Note that the desired distribution, $P_1(x)$, is a peaked distribution centered at zero, so half of the probability mass will be less than zero and half will be greater than zero. We can thus break-up our inverse of the CDF into two cases.

Case 1: If our random uniform sample $z \in [0, 1/2]$, then our resulting x should be non-positive. In this case we can write $P_1(x)$ as

$$P_1^-(x) = \frac{\exp(x)}{2} \tag{A.5}$$

We can now easily solve for $f(z) = h^{-1}(z)$ where

$$z = h(x) = \frac{1}{2}\int_{-\infty}^{x} \exp(\hat{x})d\hat{x}. \tag{A.6}$$

Solving the integral and inverting gives

$$x = \ln(2z). \tag{A.7}$$

Case 2: $z \in [1/2, 1]$. In this case, x, our resulting sample from $P_1(x)$ should be non-negative. Thus, we can write $P_1(x)$ as

$$P_1^+(x) = \frac{\exp(-x)}{2} \tag{A.8}$$

We can now solve for $f(z) = h^{-1}(z)$ where this time

$$\begin{aligned} z = h(x) &= \frac{1}{2}\int_{-\infty}^{x} \exp(-\hat{x}) && \text{(A.9)} \\ &= \frac{1}{2} + \frac{1}{2}\int_{0}^{x} \exp(-\hat{x})d\hat{x}. && \text{(A.10)} \end{aligned}$$

Solving the and inverting gives

$$x = -\ln(2 - 2z). \tag{A.11}$$

In summary, to sample from $P_1(x) = \frac{\exp(-|x|)}{2}$ we first draw $z \sim [0, 1]$. Then we return

$$x = \begin{cases} \ln(2z), & \text{for } z < 1/2 \\ -\ln(2 - 2z), & \text{otherwise} \end{cases} \tag{A.12}$$

Using d samples from $P_1(x)$ and then normalize the resulting sample gives us a way to uniformly sample the L1-norm unit sphere. This method summarized in Algorithm 7 for uniformly sampling from the L1 unit sphere.

Algorithm 7 L1-Norm Unit Ball Sampling in $\mathbb{R}^d$

input: d ▷ number of dimensions
for $i = 1 : d$ **do**
    $z \sim U(0, 1)$
    if $z \leq 0.5$ **then**
        $X_i = \ln(2z)$
    else
        $X_i = -\ln(2 - 2z)$
$\mathbf{X} \leftarrow (X_1, \ldots, X_d) / \sum_{i=1}^{d} |X_i|$
return $\mathbf{X}$

Algorithm 8 L1-Norm Unit Ball Walk

input: $w \in \mathbb{R}^d$, stepSize ▷ initial weight vector
for each pair of dimensions $(i, j) : i, j = 1, \ldots, d$ **do**
    direction $\leftarrow$ random('clockwise', 'counterclockwise')
    if $w[i]$ **is not** 0 **or** $w[j]$ **is not** 0 **then**
        $w[i], w[j] \leftarrow$ L1ManifoldStep($w[i]$, $w[j]$, direction, η)
return w

A.2.2 MCMC implementation details

Our MCMC implementation of BIRL ensures that each proposal step remains on the L1-norm unit ball. We use Algorithm 8 to generate a proposal by taking a small step along each pair of axis while staying on the L1-norm unit ball. For each pair of axis we use Algorithm 9 to step along the manifold defined by the two axis.

In all of our grid world experiments we use stepSize = 0.01 for the L1-Norm Unit Ball Walk described in Algorithm 8. We run MCMC for 10000 steps using a burn-in of 100 samples and only using every 20th sample to avoid autocorrelation effects.

We found that the BIRL likelihood can be sensitive to data imbalance if the demonstrations contain some state-action pairs much more frequently than others. To ameliorate this problem, we remove duplicate state-action pairs from the demonstrations.

Algorithm 9 L1ManifoldStep

input: w1, w2 $\in \mathbb{R}$, direction$\in$ {clockwise, counterclockwise}, stepSize $\in \mathbb{R}$
slack = w1 + w2
clockwisePos = [“+ +”,“+ -”,“- -”,“- +”], counterclockwisePos = [“+ +”,“- +”,“- -”,“+ -”]
clockwiseDir = [+1,-1,+1,-1], counterclockwiseDir = [-1,+1,-1,+1]
sign1 = (w1 $\geq$ 0), sign2 = (w2 $\geq$ 0) ▷ find starting quadrant
if sign1 **and** sign2 **then**
    cyclePos = ”+ +”
else if sign1 **and not** sign2 **then**
    cyclePos = ”+ -”;
else if not sign1 **and** sign2 **then**
    cyclePos = ”- +”
else
    cyclePos = ”- -”
if direction **is** ”clockwise” **then** ▷ find direction to change magnitude of w1
    cycleIndx = clockwisePos.indexOf(cyclePos)
else
    cycleIndx = counterclockwisePos.indexOf(cyclePos)
stepRemaining = stepSize
while stepRemaining $>$ 0 **do** ▷ step along 1-D manifold of L1-unit ball in 2-D
    if direction **is** ”clockwise” **then**
        cycleDir = clockwiseDir[cycleIndx]
    else
        cycleDir = counterclockwiseDir[cycleIndx]
    maxStep = stepRemaining
    if cycleDir **is** 1 **and** (|w1| + cycleDir * stepRemaining) $>$ slack **then**
        maxStep = slack - |w1|
        cycleIndx = mod(cycleIndx + 1, 4)
    else if cycleDir **is** -1 **and** (|w1| + cycleDir * stepRemaining) $<$ 0 **then**
        maxStep = |w1|
        cycleIndx = mod(cycleIndx + 1, 4)
    w1 = |w1| + cycleDir * maxStep
    w2 = |w2| - cycleDir * maxStep
    stepRemaining = stepRemaining - maxStep
if randDir **is** ”clockwise” **then** ▷ determine correct signs based on final quadrant
    cyclePos = clockwisePos[cycleIndx]
else
    cyclePos = counterclockwisePos[cycleIndx]
if cyclePos **is** ”- +” **then**
    w1 = -w1
else if cyclePos **is** ”+ -” **then**
    w2 = -w2
else if cyclePos **is** ”- -” **then**
    w1 = -w1
    w2 = -w2
return w1, w2

Appendix B

Theory and Proofs for Computational Efficient Inverse Reinforcement Learning from Suboptimal Demonstrations

B.1 Extrapolating Beyond a Demonstrator

We consider the case where the reward function of the demonstrator is approximated by a linear combination of features $R(s) = w^T \phi(s)$. Note that these can be arbitrarily complex features, such as the activations of the penultimate layer of a deep neural network. The expected return of a policy when evaluated on $R(s)$ is given by

$$J(\pi|R) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] = w^T \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)\right] = w^T \Phi_\pi, \tag{B.1}$$

where Φ_π are the expected discounted feature counts that result from following the policy π.

Theorem 2. *If the estimated reward function is $\hat{R}(s) = w^T\phi(s)$ and the true reward function is $R^*(s) = \hat{R}(s) + \epsilon(s)$ for some error function $\epsilon : \mathcal{S} \to \mathbb{R}$ and $\|w\|_1 \leq 1$, then extrapolation beyond the demonstrator, i.e., $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$, is guaranteed if:*

$$J(\pi^*_{R^*}|R^*) - J(\mathcal{D}|R^*) > \epsilon_\Phi + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{B.2}$$

where $\pi^*_{R^*}$ is the optimal policy under R^*, $\epsilon_\Phi = \|\Phi_{\pi^*} - \Phi_{\hat{\pi}}\|_\infty$ and $\|\epsilon\|_\infty = \sup\{\, |\epsilon(s)| : s \in \mathcal{S}\,\}$.

Proof. In order for extrapolation to be possible, the demonstrator must perform worse than $\hat{\pi}$, the policy learned via IRL, when evaluated under the true reward function. We define $\delta = J(\pi^*_{R^*}|R^*) - J(\mathcal{D}|R^*)$ as the optimality gap between the demonstrator and the optimal policy under the true reward function. We want to ensure that $J(\pi^*|R^*) - J(\hat{\pi}|R^*) < \delta$.

We have

$$J(\pi^*_{R^*}|R^*) - J(\hat{\pi}|R^*) = \left|\mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)\big]\right| \tag{B.3}$$

$$= \left|\mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t (w^T\phi(s_t) + \epsilon(s_t))\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t (w^T\phi(s_t) + \epsilon(s_t))\big]\right| \tag{B.4}$$

$$= \Bigg|w^T\mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)\big] + \mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big] - w^T\mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big]\Bigg| \tag{B.5}$$

$$= \left|w^T\Phi_{\pi^*} + \mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big] - w^T\Phi_{\hat{\pi}} - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big]\right| \tag{B.6}$$

$$= \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \mathbb{E}_{\pi^*}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big] - \mathbb{E}_{\hat{\pi}}\big[\sum_{t=0}^{\infty}\gamma^t\epsilon(s_t)\big]\right| \tag{B.7}$$

$$\leq \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \big[\sum_{t=0}^{\infty}\gamma^t \sup_{s\in\mathcal{S}}\epsilon(s)\big] - \big[\sum_{t=0}^{\infty}\gamma^t \inf_{s\in\mathcal{S}}\epsilon(s)\big]\right| \tag{B.8}$$

$$= \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \big(\sup_{s\in\mathcal{S}}\epsilon(s) - \inf_{s\in\mathcal{S}}\epsilon(s)\big)\sum_{t=0}^{\infty}\gamma^t\right| \tag{B.9}$$

$$\leq \left|w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}}) + \frac{2\|\epsilon\|_\infty}{1-\gamma}\right| \tag{B.10}$$

$$\leq |w^T(\Phi_{\pi^*} - \Phi_{\hat{\pi}})| + \left|\frac{2\|\epsilon\|_\infty}{1-\gamma}\right| \tag{B.11}$$

$$\leq \|w\|_1\|\Phi_{\pi^*} - \Phi_{\hat{\pi}}\|_\infty + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{B.12}$$

$$\leq \epsilon_\Phi + \frac{2\|\epsilon\|_\infty}{1-\gamma} \tag{B.13}$$

where $\Phi_\pi = \mathbb{E}_\pi[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)]$ and line (B.12) results from Hölder's inequality. Thus, as

long as $\delta > \epsilon_\Phi - \frac{2\|\epsilon\|_\infty}{1-\gamma}$, then $J(\pi^*|R^*) - J(\hat{\pi}|R^*) < J(\pi^*|R^*) - J(\mathcal{D}|R^*)$ and thus, $J(\hat{\pi}|R^*) > J(\mathcal{D}|R^*)$. □

Theorem 1 makes the assumption that the learner and demonstrator operate in the same state-space. However, because Theorem 1 only involve differences in expected features and reward errors over state visitations, the transition dynamics or action spaces are not required to be the same between the demonstrator and the learner. While T-REX (Section 5.2) does not require demonstrator actions, our D-REX experiments (Section 5.3) use demonstrator actions to perform behavioral cloning. Future work includes use an imitation learning from observation method (Torabi et al., 2018a) to learn an initial cloned policy without requiring demonstrator actions. This would make D-REX robust to action differences between the demonstrator and learner and allow the learner to potentially learn a more efficient policy if its action space and transition dynamics are better suited to the task.

B.2 Extrapolation via ranked demonstrations

The previous results demonstrate that in order to extrapolate beyond a demonstrator, it is sufficient to have small reward approximation error. However, the following proposition, adapted with permission from (Castro et al., 2019), demonstrates that the reward functions inferred by an IRL or apprenticeship learning algorithm may be quite superficial and may not accurately represent some dimensions of the true reward function.

Proposition 1. *There exist MDPs with true reward function R^*, expert policy π_E, approximate reward function $\hat{R}$, and non-expert policies π_1 and π_2, such that*

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|R^*) \;\; \textit{and} \;\; J(\pi_1|R^*) \ll J(\pi_2|R^*) \tag{B.14}$$

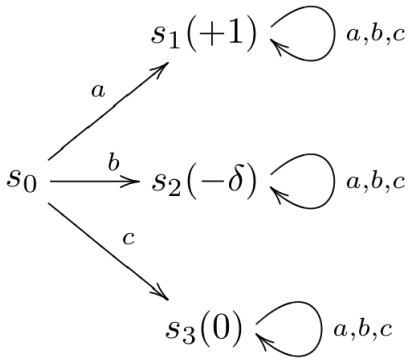
$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \;\; \textit{and} \;\; J(\pi_1|\hat{R}) = J(\pi_2|\hat{R}). \tag{B.15}$$

However, enforcing a preference ranking over trajectories, $\tau^* \succ \tau_2 \succ \tau_1$, where $\tau^* \sim \pi^*$,

$\tau_2 \sim \pi_2$, *and* $\tau_1 \sim \pi_1$, *results in a learned reward function* $\hat{R}$, *such that*

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) \;\; \textit{and} \;\; J(\pi_1|\hat{R}) < J(\pi_2|\hat{R}). \tag{B.16}$$

Proof. Consider the MDP shown below. There are three actions a, b, c, with deterministic



transitions. Each transition is labeled by the action name. The true reward received upon entering a state is indicated in parenthesis, and $\delta \gg 0$ is some arbitrary constant. Clearly, $\pi_E(s_0) = a$. Setting $\hat{R}(s_1) = 1$, $\hat{R}(s_2) = \hat{R}(s_3) = 0$, $\pi_1(s_0) = b$, and $\pi_2(s_0) = c$ provides the existence proof for Equations (B.14) and (B.15).

Enforcing the preference constraints $\tau^* \succ \tau_1 \succ \tau_2$ for $\tau^* = (s_0, a, s_1)$, $\tau_2 = (s_0, b, s_2)$, $\tau_1 = (s_0, c, s_3)$, results in a learned reward function $\hat{R}$ such that $J(\tau^*|\hat{R}) > J(\pi_2|\hat{R}) \succ J(\pi_1|\hat{R})$ which finishes the proof. □

Proposition 1 gives a simple example of when learning from an expert demonstration reveals little about the underlying reward structure of the MDP. While it is true that solving the MDP in the above example only requires knowing that state s_1 is preferable to all other states, this will likely lead to an agent assuming that both s_2 and s_3 are equally undesirable.

This may be problematic for several reasons. The first problem is that learning a reward function from demonstrations is typically used as a way to generalize to new situations—if there is a change in the initial state or transition dynamics, an agent can still

determine what actions it should take by transferring the learned reward function. However, if the learned reward function is drastically different than the true reward, this can lead to poor generalization, as would be the case if the dynamics in the above problem change and action a now causes a transition to state s_2. Another problem is that most learning from demonstration applications focus on providing *non-experts* the ability to program by example. Thus, the standard IRL approach of finding a reward function that maximizes the likelihood of the demonstrations may lead to reward functions that overfit to demonstrations and may be oblivious to important differences in rewards.

A natural way to alleviate these problems is via ranked demonstrations. Consider the problem of learning from a sequence of m demonstrated trajectories $\tau_1, \ldots, \tau_m$, ranked according to preference such that $\tau_1 \prec \tau_2, \ldots \prec \tau_m$. Using a set of strictly ranked demonstrations avoids the degenerate all-zero reward. Furthermore, ranked demonstrations provide explicit information about both what to do as well as what not to do in an environment.

B.3 Ranking Theory

IRL is an ill-posed problem due to reward ambiguity—given any policy, there are an infinite number of reward functions that make this policy optimal (Ng et al., 1999). However, it is possible to analyze the amount of reward ambiguity. In particular, if the reward is represented by a linear combination of weights, then the feasible region of all reward functions that make a policy optimal can be defined as an intersection of half-planes (Brown and Niekum, 2019b):

$$H_\pi = \bigcap_{\pi' \in \Pi} w^T(\Phi_\pi - \Phi_{\pi'}) \geq 0, \tag{B.17}$$

We define the *reward ambiguity*, $G(H_\pi)$, as the volume of this intersection of half-planes:

$$G(H_\pi) = \text{Volume}(H_\pi), \tag{B.18}$$

where we assume without loss of generality that $\|w\| \leq 1$, to ensure this volume is bounded.

We now prove that a total ranking over policies results in no more reward ambiguity than simply using the optimal policy.

Proposition 2. *Given a policy class* Π*, an optimal policy* $\pi^* \in \Pi$ *and a total ranking over* Π*, and a reward function* $R(s) = w^T\phi(s)$*, the reward ambiguity resulting from* π^* *is greater than or equal to the reward ambiguity of using a total ranking, i.e.,* $G(H^*_\pi) \geq G(H_{\text{ranked}})$.

Proof. Consider policies π_1 and π_2 where $J_{R^*}(\pi_1) \geq J_{R^*}(\pi_2)$. We can write this return inequality in terms of half-spaces as follows:

$$J_{R^*}(\pi_1) \geq J_{R^*}(\pi_2) \tag{B.19}$$

$$\iff \quad w^T\Phi_{\pi_1} \geq w^T\Phi_{\pi_2} \tag{B.20}$$

$$\iff \quad w^T(\Phi_{\pi_1} - \Phi_{\pi_2}) \geq 0 \tag{B.21}$$

defining a half-space over weight vectors.

Consider the optimal policy $\pi^*_{r^*}$. This policy induces a set of half-space constraints over all other possible policies $\pi \in \Pi$. Thus we have the following half-space constraints:

$$H_{\pi^*} = \bigcap_{\pi \in \Pi} w^T(\Phi_{\pi^*} - \Phi_\pi) \geq 0 \tag{B.22}$$

However, if we have a total ordering over Π, then we have the following intersection of half-spaces

$$H_{\text{ranked}} = \bigcap_{\pi_i \succsim \pi_j \in \Pi} w^T(\Phi_{\pi_i} - \Phi_{\pi_j}) \geq 0 \tag{B.23}$$

$$= H_{\pi^*} \cap \left(\bigcap_{\substack{\pi_i \succsim \pi_j \in \Pi \\ \pi_i \neq \pi^*}} w^T(\Phi_{\pi_i} - \Phi_{\pi_j}) \geq 0 \right). \tag{B.24}$$

Thus, $H_{\text{ranked}} \subseteq H_{\pi^*}$, and the volume of the set of feasible reward functions

induced by the total ranking is therefore less than or equal to the volume of H_{π^*}, i.e., $G(H_{\text{ranked}}) \leq G(H_{\pi^*})$.

□

B.4 Uncertainty Reduction for Random Halfspaces

In this section we seek to bound how many reward hypotheses are removed after a certain number of ranked trajectories, $\tau_i \prec \tau_j$. Our main result that we will prove is that sampling random half-space constraints causes the reward ambiguity to decrease exponentially.

We assume that the true reward function, $R^*(s) = w^{*T}\phi(s)$, is a linear combination of features $w \in \mathbb{R}^n$. Let $\mathcal{H}$ denote the set of all reward hypotheses. We assume a linear combination of features $R(s) = w^T\phi(s)$, so $\mathcal{H} = \mathbb{R}^n$. We also make the common assumption that $\|w\|_1 \leq 1$ so that $\mathcal{H}$ is bounded (Abbeel and Ng, 2004; Brown and Niekum, 2018).

Each pair of trajectories, $\tau_i \prec \tau_j$, that are ranked based on R^*, forms a half-space constraint over the true reward feature weights w^{*T}:

$$\tau_i \prec \tau_j \tag{B.25}$$

$$\Rightarrow \quad J(\tau_i|R^*) < J(\tau_j|R^*) \tag{B.26}$$

$$\Rightarrow \quad w^{*T}\Phi_{\tau_i} < w^{*T}\Phi_{\tau_j} \tag{B.27}$$

$$\Rightarrow \quad w^{*T}(\Phi_{\tau_j} - \Phi_{\tau_i}) > 0, \tag{B.28}$$

where $\Phi_\tau = \sum_{t=0}^{T} \gamma^t \phi(s_t)$ for $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$.

We define Ξ to be the set of all trajectories in the MDP. Let $\mathcal{X}$ be the set of half-space constraints that result from all ground-truth rankings over all possible trajectory pairs from Ξ, i.e.,

$$\mathcal{X} = \big\{(\Phi_{\tau_j} - \Phi_{\tau_i}) : J(\tau_j|R^*) > J(\tau_j|R^*), \tau_i, \tau_j \in \Xi\big\}. \tag{B.29}$$

We define the space of all reward feature weight vectors that are consistent with $\mathcal{X}$ as $\mathcal{R}$, where

$$\mathcal{R} = \left\{ w : w \in \bigcap_{x \in \mathcal{X}} w^T x > 0 \right\}, \tag{B.30}$$

where x is one of the half-space normal vectors from the set $\mathcal{X}$. We also define $\mathcal{J} = \mathcal{H} \setminus \mathcal{R}$, i.e., the space that gets cut by all the half-spaces defined by the normal vectors in $\mathcal{X}$. Our goal is to find a lower bound, in expectation, on how much volume is cut from $\mathcal{J}$ for k half-space constraints.

To simplify our proofs and notation, we assume there are only a finite number of reward weight hypotheses in $\mathcal{J}$. Denote j as the jth reward hypothesis in $\mathcal{J}$ and let $J = |\mathcal{J}|$ be the total number of these hypotheses. For each halfspace $x \in \mathcal{X}$, define $Z_i(x)$ as the binary indicator of whether halfspace x (assumed to be represented by its normal vector) eliminates the hypothesis $j \in \mathbb{R}^n$, i.e.

$$Z_j(x) = \begin{cases} 1 & \text{if } j^T x \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{B.31}$$

We define

$$\mathcal{T} = \{ j^{(\sum_{x \in \mathcal{X}} Z_j(x))} : j \in \mathcal{J} \} \tag{B.32}$$

to be the multiset containing all the hypotheses in $\mathcal{J}$ that are eliminated by the half-space constraints in $\mathcal{X}$. This is a multiset since we include repeats of j that are eliminated by different half-spaces in $\mathcal{X}$. We use the notation $\{a^{(y)}\}$ to denote the multiset with y copies of a. We define $T = |\mathcal{T}|$ to be the sum of the total number of reward hypotheses that are eliminated by each half-space in $\mathcal{X}$. Similarly, we define $H = |\mathcal{H}|$ and $X = |\mathcal{X}|$ to be the number of hypothesis in $\mathcal{H}$ and the number of half-spaces in $\mathcal{X}$, respectively.

Our goal is to find a lower bound on the expected number of hypotheses that are removed given k half-space constraints that result from k pairwise trajectory preferences.

Note that we can define T as

$$T = \sum_{j \in J} \sum_{x \in X} Z_j(x). \tag{B.33}$$

We can derive the expected number of unique hypotheses in $\mathcal{J}$ that are eliminated by the first half-space as follows:

$$\begin{aligned}
\mathbb{E}[\sum_{j \in \mathcal{J}} Z_j(x) | x \sim \mathcal{U}(\mathcal{X})] &= \sum_{j \in \mathcal{J}} \mathbb{E}_{x \in \mathcal{X}}[Z_j(x)] && \text{(B.34)} \\
&= \sum_{j \in \mathcal{J}} \sum_{x \in X} p(x) Z_j(x) && \text{(B.35)} \\
&= \sum_{j \in \mathcal{J}} \sum_{x \in \mathcal{X}} \frac{1}{X} Z_j(x) && \text{(B.36)} \\
&= \frac{1}{X} \sum_{j \in \mathcal{J}} \sum_{x \in \mathcal{X}} Z_j(x) && \text{(B.37)} \\
&= \frac{T}{X} && \text{(B.38)}
\end{aligned}$$

where $\mathcal{U}(\mathcal{X})$ represents a uniform distribution over $\mathcal{X}$. There are T hypotheses left to eliminate and X halfspaces left to choose from. Once we choose all of them we have eliminated all of $\mathcal{T}$, so, in expectation, each half-space elminates T/X of the hypotheses in $\mathcal{T}$. We can now consider T to be the total number of hypotheses that remain to be eliminated after already enforcing some number of half-space constraints, and the same reasoning applies. We will take advantage of this later to form a recurrence relation over the elements left in $\mathcal{J}$, i.e. the reward function hypotheses that are false, but have not yet been eliminated by a pairwise trajectory ranking.

When selecting a half-space from $\mathcal{X}$ uniformly, we eliminate T/X unique hypotheses in $\mathcal{J}$ in expectation. If we assume that each $j \in \mathcal{J}$ is distributed uniformly across the half-spaces, then on average, there are T/J copies of a hypothesis j in $\mathcal{T}$. This can be

shown formally as follows:

$$\mathbb{E}_{j\in\mathcal{J}}[\sum_{x\in\mathcal{X}} Z_j(x)] = \sum_{j\in\mathcal{J}} p(j)\Big[\sum_{x\in\mathcal{X}} Z_j(x)\Big] \tag{B.39}$$

$$= \frac{1}{J}\sum_{j\in\mathcal{J}}\sum_{x\in\mathcal{X}} Z_j(x) \tag{B.40}$$

$$= \frac{T}{J}. \tag{B.41}$$

Thus, selecting a half-space uniformly at random eliminates T/X unique hypotheses from $\mathcal{J}$; however, there are T/J copies of each hypothesis on average, so a random half-space eliminates $\frac{T^2}{JX}$ total hypotheses from $\mathcal{T}$ on average.

Recurrence relation The above analysis results in the following system of recurrence relations where J_k is the number of unique hypotheses that have not yet been eliminated after k half-spaces constraints, T_k is the sum of the total hypotheses (including repeats) that have not yet been eliminated the first k half-spaces, and X is the total number of half-space constraints.

$$J_0 = J \tag{B.42}$$

$$T_0 = T \tag{B.43}$$

$$J_{k+1} = J_k - \frac{T_k}{X-k} \tag{B.44}$$

$$T_{k+1} = T_k - \frac{T_k^2}{J_k\cdot(X-k)} \tag{B.45}$$

To simplify the analysis we make the following assumptions. First we assume that $\mathcal{R}$ is small compared to the full space of $\mathcal{H}$. Thus, $\mathcal{H}\approx\mathcal{J}$. Since each $x\in\mathcal{X}$ covers $H/2$ hypotheses, then each $x\in\mathcal{X}$ covers approximately $J/2$ hypotheses. Summing over all halfspaces, we have that $T_0\approx\sum_{x\in\mathcal{X}} J/2 = XJ/2$. Furthermore, if X, the number of possible half-spaces, is large, then $X-k\approx X$.

Given these assumptions we now have the following simplified system of recurrence relations:

$$J_0 = J \tag{B.46}$$

$$T_0 = \frac{J_0 X}{2} \tag{B.47}$$

$$J_{k+1} = J_k - \frac{T_k}{X} \tag{B.48}$$

$$T_{k+1} = T_k - \frac{T_k^2}{J_k X} \tag{B.49}$$

where T_k and J_k are the number of elements in $\mathcal{T}$ and $\mathcal{J}$ after k half-spaces constraints have been applied.

The solution to these recurrences is:

$$J_k = \frac{J_0}{2^k}, \tag{B.50}$$

$$T_k = \frac{J_0 X}{2^{k+1}}. \tag{B.51}$$

and can be verified by plugging the solution into the above equations and checking that it satisfies the base cases and recurrence relations.

We can now bound the number of half-space constraints (the number of ranked random trajectory pairs) that are needed to get $J = |\mathcal{J}|$ down to some level ϵ.

Theorem 3. *To reduce the volume of $\mathcal{J}$ such that $J_k = \epsilon$, then it suffices to have k random half-space constraints, where*

$$k = \log_2 \frac{J_0}{\epsilon} \tag{B.52}$$

Proof. As shown above, we have that

$$J_k = \frac{J_0}{2^k}. \tag{B.53}$$

Solving for k and substituting ϵ for J_k we get

$$k = \log_2 \frac{J_0}{\epsilon} \tag{B.54}$$

□

This means that J_k decreases exponentially with k, meaning we only need to sample a logarithmic number of half-space constraints to remove a large portion of the possible reward hypotheses.

Corollary 2. *To reduce J by $x\%$ it suffices to have*

$$k = \log_2 \left(\frac{1}{1 - x/100} \right) \tag{B.55}$$

random half-space constraints.

Proof. We want

$$\frac{x}{100} = \frac{J_0 - \frac{J_0}{2^k}}{J_0} = 1 - \frac{1}{2^k} \tag{B.56}$$

Solving for k results in

$$k = \log_2 \left(\frac{1}{1 - x/100} \right). \tag{B.57}$$

□

B.5 Noise Injection Theory

Let ϵ be the probability of taking a random action. We assume that $\mathcal{A}$ is finite, and assume a finite horizon MDP with a horizon of T.

B.5.1 Optimal policy

We start by analyzing what happens when we inject noise into an optimal policy, π^*. We will inject noise using the ϵ-greedy policy defined in Equation (D.2). We define p_ϵ to be

the probability of taking an suboptimal action. For simplicity, we assume that there is only one optimal action at each state and that the states visited are independent of each other. This simplifies the analysis, but ignores the problems of compounding error that occur in sequential decision making tasks (Ross and Bagnell, 2010). We will address the impact of compounding errors in Section B.5.3.

Given the above assumptions we have

$$p_\epsilon = \frac{\epsilon(|A| - 1)}{|A|}, \tag{B.58}$$

where A is the set of actions in the MDP.

If we define X to be the random variable representing the number of suboptimal actions in a trajectory of length T, then we can then analyze the number of suboptimal actions in the epsilon greedy policy $\pi^*(\cdot|\epsilon)$ using a binomial distribution with T trials and probability $p = p_\epsilon$.

This gives us $\mathbb{E}[X] = Tp_\epsilon$. If we assume that $|A|$ is large, then $p_\epsilon \approx \epsilon$ and we have $\mathbb{E}[X] = T\epsilon$. Thus, as ϵ goes from 0 to 1, the expected number of suboptimal actions in the policy interpolates between 0 and T. Furthermore, a standard Hoeffding bound shows that for large T, i.e., long trajectories, the empirical number of suboptimal actions will concentrate tightly around the mean $T\epsilon$. Thus, as ϵ goes from 0 to 1, with high probability the empirical number of suboptimal actions will interpolate between 0 and T.

B.5.2 Suboptimal cloned policy

Suppose that instead of injecting noise into the optimal policy, we now inject noise into a policy, π_{BC}, that is cloned from suboptimal demonstrations. Define β to be the probability that π_{BC} takes the optimal action in a state. Note that here we also simplify the analysis by assuming that β is a constant that does not depend on the current state. We define p_{BC}^{opt} to be the probability of taking a suboptimal action in state s if π_{BC} would take the optimal action in that state. Similarly, we define p_{BC}^{sub} to be the probability of taking a suboptimal action in

state s if π_{BC} would take a suboptimal action in that state.

As before we define p_ϵ to be the probability that $\pi_{\mathrm{BC}}(\cdot|\epsilon)$ takes a suboptimal action. Thus, we have that

$$p_\epsilon = \beta p_{\mathrm{BC}}^{opt} + (1-\beta) p_{\mathrm{BC}}^{sub}, \tag{B.59}$$

where

$$p_{\mathrm{BC}}^{opt} = \frac{\epsilon(|A|-1)}{|A|} \tag{B.60}$$

and

$$p_{\mathrm{BC}}^{sub} = 1 - \epsilon + \frac{\epsilon(|A|-1)}{|A|}. \tag{B.61}$$

Thus, we have

$$\begin{aligned} p_\epsilon &= \beta \cdot \frac{\epsilon(|A|-1)}{|A|} + (1-\beta)\cdot(1-\epsilon+\frac{\epsilon(|A|-1)}{|A|}) && \text{(B.62)} \\ &= (1-\beta)(1-\epsilon) + \frac{\epsilon(|A|-1)}{|A|}. && \text{(B.63)} \end{aligned}$$

As expected, when $\beta = 1$, then the first term is zero and we have the same result as for an optimal policy. Also, if $\epsilon = 1$, then all actions are random and we get the same result as our above analysis which assumed noise injection into an optimal policy. This is because the optimality of the cloned policy does not affect the analysis if all actions are random.

If we assume that $|A|$ is large, then we have $(|A|-1)/|A| \approx 1$ and we have

$$p_\epsilon = 1 - \beta(1-\epsilon). \tag{B.64}$$

The expected number of suboptimal actions in the epsilon greedy cloned policy becomes

$$\mathbb{E}[X] = T \cdot (1 - \beta(1-\epsilon)). \tag{B.65}$$

Thus, with no noise injection ($\epsilon = 0$), we expect the cloned policy to make $(1-\beta)T$ suboptimal actions. For large enough A and T, as ϵ goes from 0 to 1 the empirical number

of suboptimal actions will, with high probability, interpolate between $(1-\beta)T$ and T.

B.5.3 Compounding errors

The above analysis has ignored the compounding errors that are accumulated when running a behavioral cloned policy in a sequential decision making task. Using the same analysis as Ross and Bagnell (2010), we get the following result that proves that the performance of the cloned policy with noise injection ϵ has an increasingly large performance gap from the expert as ϵ goes from 0 to 1.

Corollary 3. *Given a cloned policy π_{BC} from expert demonstrations, if the cloned policy makes a mistake with probability $1-\beta$, then*

$$J(\pi_{\mathrm{BC}}(\cdot|\epsilon)) \leq J(\pi^*) + T^2(1-\beta(1-\epsilon)) \tag{B.66}$$

where T is the time-horizon of the MDP.

Proof. As demonstrated above, given $|A|$ sufficiently large, we can define p_ϵ, the probability of taking a suboptimal action when following $\pi_{BC}(\cdot|\epsilon)$ is as follows:

$$\begin{aligned} p_\epsilon &= \beta\frac{\epsilon(|A|-1)}{|A|} + (1-\beta)(1-\epsilon+\frac{\epsilon(|A|-1)}{|A|}) && \text{(B.67)} \\ &= (1-\beta)(1-\epsilon) + \frac{\epsilon(|A|-1)}{|A|} && \text{(B.68)} \\ &\approx 1-\beta(1-\epsilon). && \text{(B.69)} \end{aligned}$$

The inequality then follows from the proof of Theorem 2.1 in (Ross and Bagnell, 2010). □

Appendix C

Supplementary Materials for Trajectory-Ranked Reward Extrapolation

C.1 Code and Videos

Code as well as supplemental videos are available at the project website:
https://github.com/hiwonjoon/ICML2019-TREX.

C.2 T-REX Results on the MuJoCo Domain

C.2.1 Policy visualization

We visualized the T-REX-learned policy for HalfCheetah in Figure C.1. Visualizing the demonstrations from different stages shows the specific way the policy evolves over time; an agent learns to crawl first and then begins to attempt to walk in an upright position. The T-REX policy learned from the highly suboptimal Stage 1 demonstrations results in a similar-style crawling gait; however, T-REX captures some of the intent behind the demonstration

(a) Stage 1

(b) Stage 2

(c) Stage 3

Figure C.1: HalfCheetah policy visualization. For each subplot, (top) is the best given demonstration policy in a stage, and (bottom) is the trained policy with a T-REX reward function.

and is able to optimize a gait that resembles the demonstrator but with increased speed, resulting in a better-than-demonstrator policy. Similarly, given demonstrations from Stage 2, which are still highly suboptimal, T-REX learns a policy that resembles the gait of the best demonstration, but is able to optimize and partially stabilize this gait. Finally, given demonstrations from Stage 3, which are still suboptimal, T-REX is able to learn a near-optimal gait.

C.3 Behavioral Cloning from Observation

To build the inverse transition models used by BCO Torabi et al. (2018a) we used 20,000 steps of a random policy to collect transitions with labeled states. We used the Adam optimizer with learning rate 0.0001 and L2 regularization of 0.0001. We used the DQN architecture Mnih et al. (2015) for the classification network, using the same architecture to predict actions given state transitions as well as predict actions given states. When predicting $P(a|s_t, s_{t+1})$, we concatenate the state vectors obtaining an 8x84x84 input consisting of two 4x84x84 frames representing s_t and s_{t+1}.

We give both T-REX and BCO the full set of demonstrations. We tried to improve the performance of BCO by running behavioral cloning only on the best $X\%$ of the demonstrations, but were unable to find a parameter setting that performed better than $X = 100$, likely due to a lack of training data when using very few demonstrations.

C.4 Atari reward learning details

We used the OpenAI Baselines implementation of PPO with default hyperparameters. We ran all of our experiments on an NVIDIA TITAN V GPU. We used 9 parallel workers when running PPO.

When learning and predicting rewards, we mask the score and number of lives left for all games. We did this to avoid having the network learn to only look at the score and recognize, say, the number of significant digits, etc. We additionally masked the sector number and number of enemy ships left on Beam Rider. We masked the bottom half of the dashboard for Enduro to mask the position of the car in the race. We masked the number of divers found and the oxygen meter for Seaquest. We masked the power level and inventory for Hero.

To train the reward network for Enduro, we randomly downsampled full trajectories. To create a training set we repeatedly randomly select two full demonstrations, then randomly cropped between 0 and 5 of the initial frames from each trajectory and then downsampled both trajectories by only keeping every xth frame where x is randomly chosen between 3 and 6. We selected 2,000 randomly downsampled demonstrations and trained the reward network for 10,000 steps of Adam with a learning rate of 5e-5.

C.5 Comparison to active reward learning

In this section, we examine the ability of prior work on active preference learning to exceed the performance of the demonstrator. In Table C.1, we denote the results that surpass the best demonstration with an asterisk (*). DQfD+A only surpasses the demonstrator in 3 out of 9 games tested, even with thousands of active queries. Note that DQfD+A extends the original DQfD algorithm Hester et al. (2018), which uses demonstrations combined with RL on ground-truth rewards, yet is only able to surpass the best demonstration in 14 out of 41 Atari games. In contrast, we are able to leverage only 12 ranked demos to achieve better-than-demonstrator performance on 7 out of 8 games tested, without requiring access to true rewards or access to thousands of active queries from an oracle.

Ibarz et al. (2018) combine Deep Q-learning from demonstrations and active preference queries (DQfD+A). DQfD+A uses demonstrations consisting of (s_t, a_t, s_{t+1})-tuples to initialize a policy using DQfD Hester et al. (2018). The algorithm then uses the active preference learning algorithm of Christiano et al. (2017) to refine the inferred reward function and initial policy learned from demonstrations. The first two columns of Table C.1 compare the demonstration quality given to DQfD+A and T-REX. While our results make use of more demonstrations (12 for T-REX versus 4–7 for DQfD+A), our demonstrations are typically orders of magnitude worse than the demonstrations used by DQfD+A: on average the demonstrations given to DQfD+A are 38 times better than those used by T-REX. However, despite this large gap in the performance of the demonstrations, T-REX surpasses the performance of DQfD+A on Q*Bert, and Seaquest. We achieve these results using 12 ranked demonstrations. This requires only 66 comparisons ($n \cdot (n-1)/2$) by the demonstrator. In comparison, the DQfD+A results used 3,400 preference labels obtained during policy training using ground-truth rewards.

C.6 Human Demonstrations and Rankings

C.6.1 Human demonstrations

We used the Atari Grand Challenge data set Kurin et al. (2017) to collect actual human demonstrations for five Atari games. We used the ground truth returns in the Atari Grand Challenge data set to rank demonstrations. To generate demonstrations we removed duplicate demonstrations (human demonstrations that achieved the same score). We then sorted the remaining demonstrations based on ground truth return and selected 12 of these demonstrations to form our training set. We ran T-REX using the same hyperparameters as described above.

The resulting performance of T-REX is shown in Table 5.2. T-REX is able to outperform the best human demonstration on Q*bert, Space Invaders, and Video Pinball; however, it is not able to learn a good control policy for Montezuma's Revenge or Ms Pacman. These games require maze navigation and balancing different objectives, such as collecting objects and avoiding enemies. This matches our results in the main text that show that T-REX is unable to learn a policy for playing Hero, a similar maze navigation task with multiple objectives such as blowing up walls, rescuing people, and destroying enemies. Extending T-REX to work in these types of settings is an interesting area of future work.

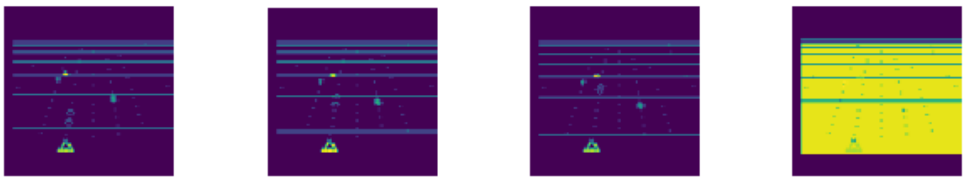
C.7 Atari Reward Visualizations

We generated attention maps for the learned rewards for the Atari domains. We use the method proposed by Greydanus et al. (2018), which takes a stack of 4 frames and passes a 3x3 mask over each of the frames with a stride of 1. The mask is set to be the default background color for each game. For each masked 3x3 region, we compute the absolute difference in predicted reward when the 3x3 region is not masked and when it is masked. This allows us to measure the influence of different regions of the image on the predicted reward. The sum total of absolute changes in reward for each pixel is used to generate an

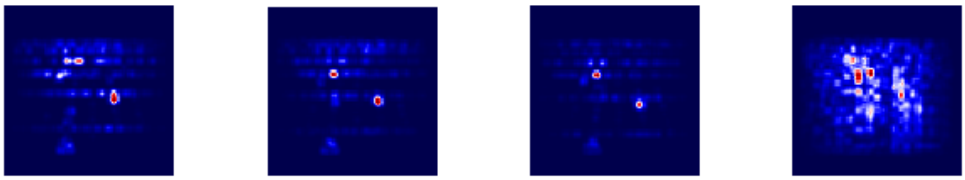
attention heatmap. We used the trajectories shown in the extrapolation plots in Figure 4 of the main text and performed a search using the learned reward function to find the observations with minimum and maximum predicted reward. We show the minimum and maximum observations (stacks of four frames) along with the attention heatmaps across all four stacked frames for the learned reward functions in figures C.2–C.9. The reward function visualizations suggest that our networks are learning relevant features of the reward function.

Table C.1: Best demonstrations and average performance of learned policies for T-REX (ours) and DQfD with active preference learning (DQfD+A) (see Ibarz et al. (2018) Appendix A.2 and G). Results for T-REX are the best performance over 3 random seeds averaged over 30 trials. Results that exceed the best demonstration are marked with an asterisk (*). Note that T-REX requires at most only 66 pair-wise preference labels ($n(n-1)/2$ for $n = 12$ demonstrations), whereas DQfD+A uses between 4–7 demonstrations along with 3.4K labels queried during policy learning. DQfD+A requires action labels on the demonstrations, whereas T-REX learns from observation.

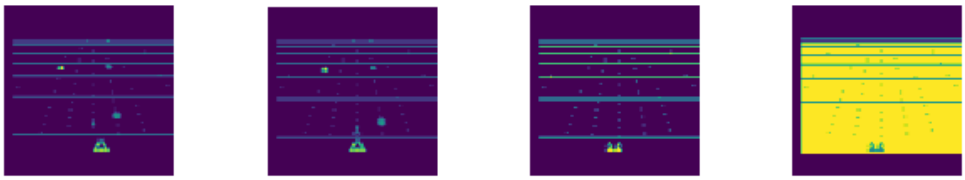
	Best Demonstration Received		Average Algorithm Performance	
Game	DQfD+A	T-REX	DQfD+A	T-REX
Beam Rider	19,844	1,188	4,100	*3,335.7
Breakout	79	33	*85	*221.3
Enduro	803	84	*1200	*586.8
Hero	99,320	13,235	35,000	0.0
Montezuma's Revenge	34,900	-	3,000	-
Pong	0	-6	*19	*-2.0
Private Eye	74,456	-	52,000	-
Q*bert	99,450	800	14,000	*32,345.8
Seaquest	101,120	600	500	*747.3
Space invaders	-	600	-	*1,032.5



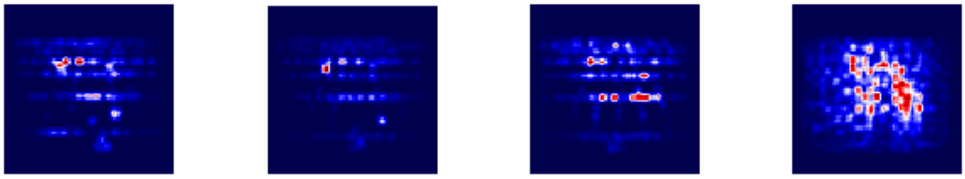
(a) Beam Rider observation with maximum predicted reward



(b) Beam Rider reward model attention on maximum predicted reward



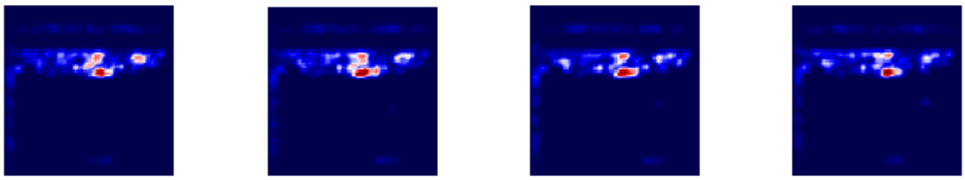
(c) Beam Rider observation with minimum predicted reward



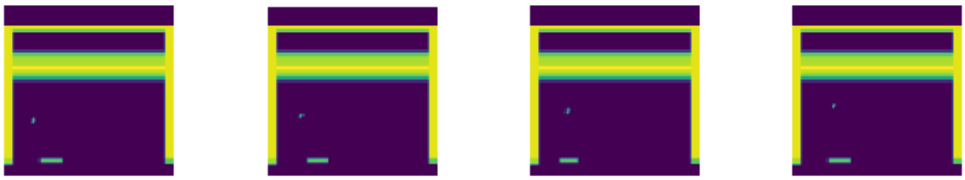
(d) Beam Rider reward model attention on minimum predicted reward

Figure C.2: Maximum and minimum predicted observations and corresponding attention maps for Beam Rider. The observation with the maximum predicted reward shows successfully destroying an enemy ship, with the network paying attention to the oncoming enemy ships and the shot that was fired to destroy the enemy ship. The observation with minimum predicted reward shows an enemy shot that destroys the player's ship and causes the player to lose a life. The network attends most strongly to the enemy ships but also to the incoming shot.

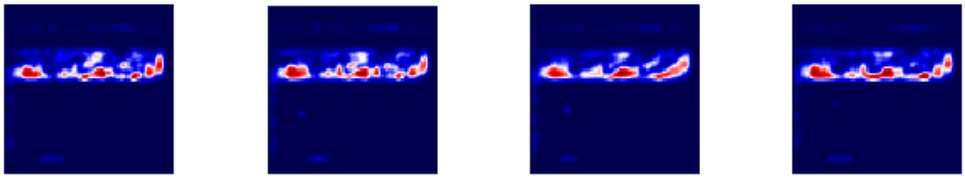
(a) Breakout observation with maximum predicted reward



(b) Breakout reward model attention on maximum predicted reward



(c) Breakout observation with minimum predicted reward



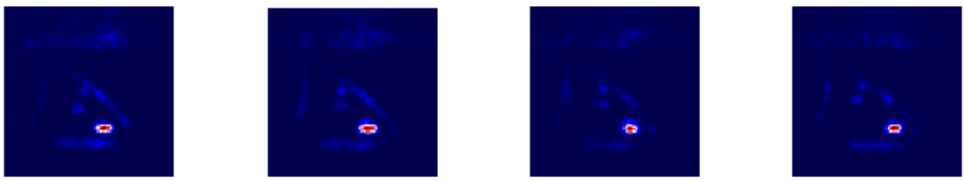
(d) Breakout reward model attention on minimum predicted reward

Figure C.3: Maximum and minimum predicted observations and corresponding attention maps for Breakout. The observation with maximum predicted reward shows many of the bricks destroyed with the ball on its way to hit another brick. The network has learned to put most of the reward weight on the remaining bricks with some attention on the ball and paddle. The observation with minimum predicted reward is an observation where none of the bricks have been destroyed. The network attention is focused on the bottom layers of bricks.

(a) Enduro observation with maximum predicted reward

(b) Enduro reward model attention on maximum predicted reward

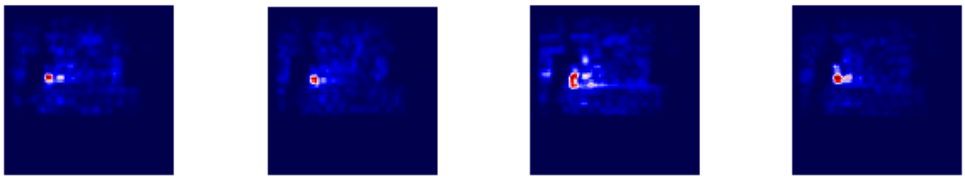
(c) Enduro observation with minimum predicted reward



(d) Enduro reward model attention on minimum predicted reward

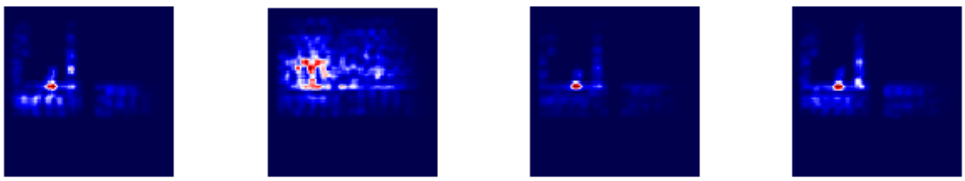
Figure C.4: Maximum and minimum predicted observations and corresponding attention maps for Enduro. The observation with maximum predicted reward shows the car passing to the right of another car. The network has learned to put attention on the controlled car as well as the sides of the road with some attention on the car being passed and on the odometer. The observation with minimum predicted reward shows the controlled car falling behind other racers, with attention on the other cars, the odometer, and the controlled car.

(a) Hero observation with maximum predicted reward



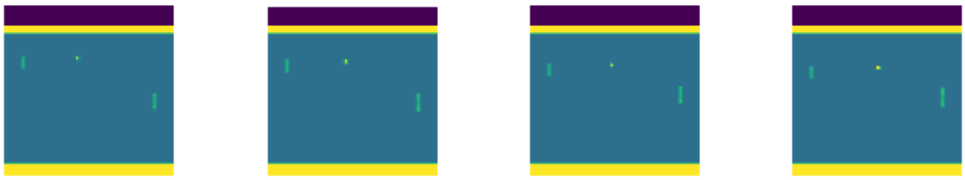
(b) Hero reward model attention on maximum predicted reward

(c) Hero observation with minimum predicted reward

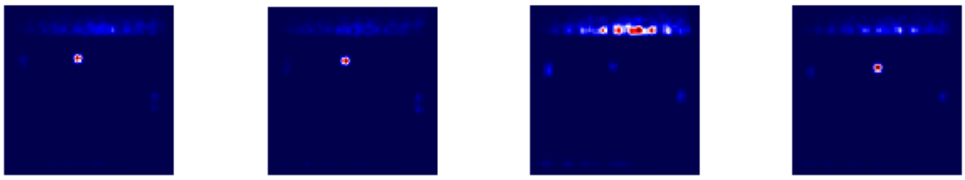


(d) Hero reward model attention on minimum predicted reward

Figure C.5: Maximum and minimum predicted observations and corresponding attention maps for Hero. The observation with maximum predicted reward is difficult to interpret, but shows the network attending to the controllable character and the shape of the surrounding maze. The observation with minimum predicted reward shows the agent setting off a bomb that kills the main character rather than the wall. The learned reward function attends to the controllable character, the explosion and the wall that was not destroyed.

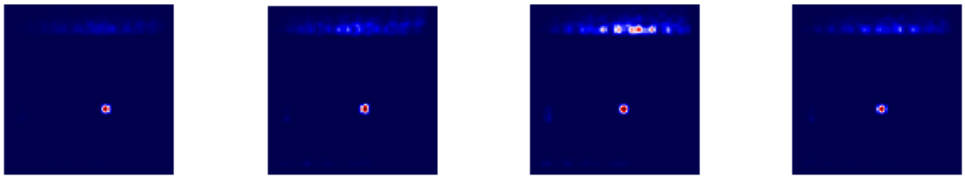


(a) Pong observation with maximum predicted reward



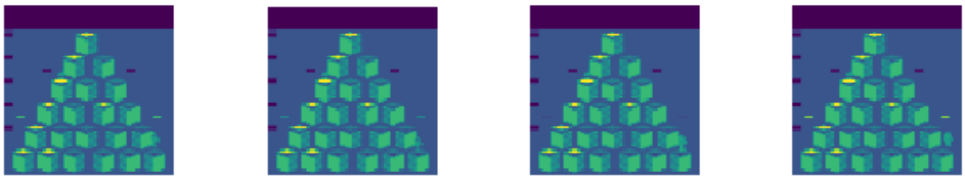
(b) Pong reward model attention on maximum predicted reward

(c) Pong observation with minimum predicted reward

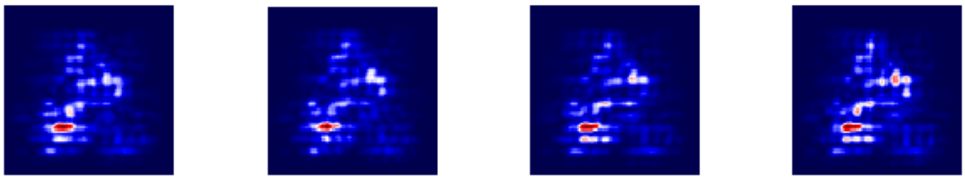


(d) Pong reward model attention on minimum predicted reward

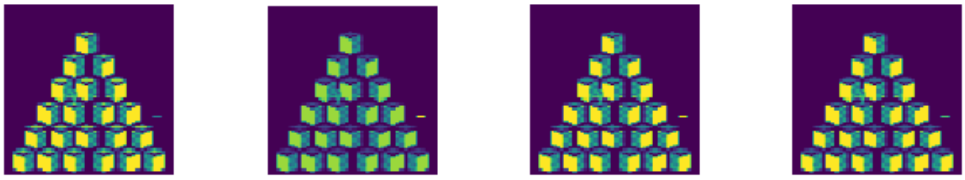
Figure C.6: Maximum and minimum predicted observations and corresponding attention maps for Pong. The network mainly attends to the ball, with some attention on the paddles.



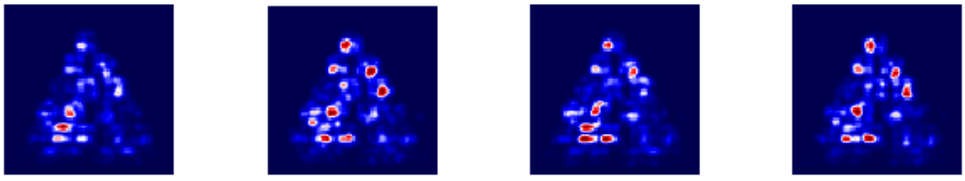
(a) Q*bert observation with maximum predicted reward



(b) Q*bert reward model attention on maximum predicted reward

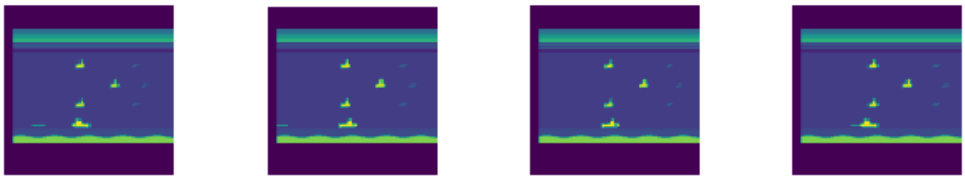


(c) Q*bert observation with minimum predicted reward

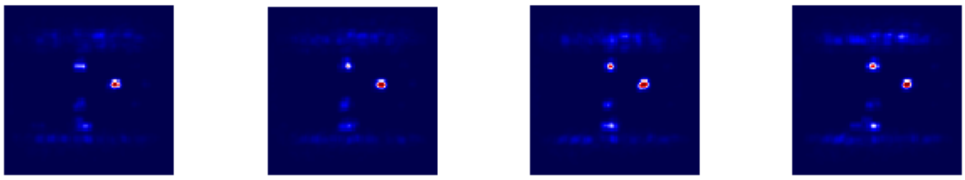


(d) Q*bert reward model attention on minimum predicted reward

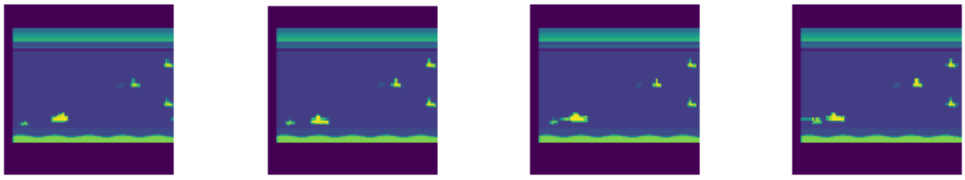
Figure C.7: Maximum and minimum predicted observations and corresponding attention maps for Q*bert. The observation for the maximum predicted reward shows an observation from the second level of the game where stairs change color from yellow to blue. The observation for the minimum predicted reward is less interpretable. The network attention is focused on the different stairs, but it is difficult to attribute any semantics to the attention maps.



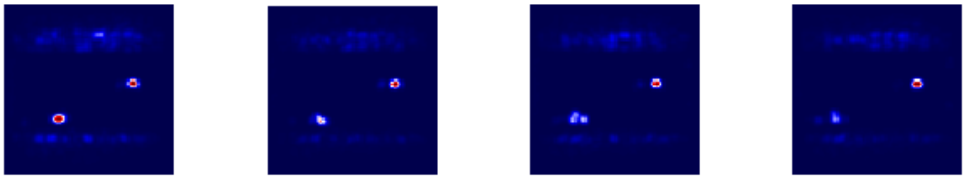
(a) Seaquest observation with maximum predicted reward



(b) Seaquest reward model attention on maximum predicted reward



(c) Seaquest observation with minimum predicted reward

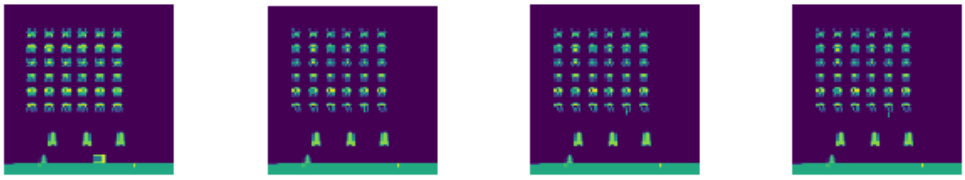


(d) Seaquest reward model attention on minimum predicted reward

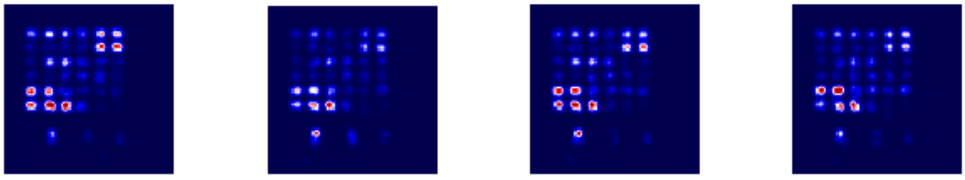
Figure C.8: Maximum and minimum predicted observations and corresponding attention maps for Seaquest. The observation with maximum predicted reward shows the submarine in a relatively safe area with no immediate threats. The observation with minimum predicted reward shows an enemy that is about to hit the submarine—the submarine fires a shot, but misses. The attention maps show that the network focuses on the nearby enemies and also on the controlled submarine.

(a) Space Invaders observation with maximum predicted reward

(b) Space Invaders reward model attention on maximum predicted reward



(c) Space Invaders observation with minimum predicted reward



(d) Space Invaders reward model attention on minimum predicted reward

Figure C.9: Maximum and minimum predicted observations and corresponding attention maps for Space Invaders. The observation with maximum predicted reward shows an observation where all the aliens have been successfully destroyed and the protective barriers are still intact. Note that the agent never observed a demonstration that successfully destroyed all the aliens. The attention map shows that the learned reward function is focused on the barriers, but does not attend to the location of the controlled ship. The observation with minimum predicted reward shows the very start of a game with all aliens still alive. The network attends to the aliens and barriers, with higher weight on the aliens and barrier closest to the space ship.

Appendix D

Supplementary Materials for Disturbance-Based Reward Extrapolation

D.1 D-REX Details

Code and videos can be found at our project webpage:
https://dsbrown1331.github.io/CoRL2019-DREX/.

D.1.1 Demonstrations

To create the demonstrations, we used a partially trained Proximal Policy Optimization (PPO) agent that was checkpointed every 5 optimization steps (corresponds to 10,240 simulation steps) for MuJoCo experiments and 50 optimization steps (corresponds to 51,200 simulation steps) for Atari experiments. To simulate suboptimal demonstrations, we selected demonstration checkpoints such that they resulted in an average performance that was significantly better than random play, but also significantly lower than the maximum performance achieved by PPO when trained to convergence on the ground-truth reward. All

checkpoints are included in the source code included in the supplemental materials.

D.1.2 Behavioral cloning

MuJoCo experiments We generated a trajectory of length 1,000, and the given 1,000 pairs of data is used for training. The policy network is optimized with L_2 loss for 10,000 iterations using Adam optimizer with a learning rate of 0.001 and a minibatch size of 128. Weight decay regularization is also applied in addition to regular loss term with a coefficient of 0.001. A multi-layer perceptron (MLP) having 4 layers and 256 units in the middle is used to parameterize a policy.

Atari experiments We used the state-action pairs from the 10 demonstrations and partitioned them into an 80% train 20% validation split. We used the Nature DQN network architecture and trained the imitation policy using Adam with a learning rate of 0.0001 and a minibatch size of 32. The state consists of four stacked frames which are normalized to have a value between 0 and 1, and the scores in the game scene are masked as it is done in Brown et al. (2019b). We used the validation set for early stopping. In particular, after every 1000 updates on the training data we fully calculated the validation error of the current model. We trained the imitation policy until the validation loss failed to improve for 6 consecutive calculations of the validation error.

D.1.3 Synthetic rankings

We then used the cloned policy and generated 100 synthetic demonstrations for different noise levels. For the MuJoCo experiments, we used 20 different noise levels evenly spaced over the interval $[0.0, 1.0)$ and generated 5 trajectories for each level.

For the Atari experiments, we used the following noise degradation schedule $\mathcal{E} = (1.0, 0.75, 0.5, 0.25, 0.02)$ and generated $K = 20$ trajectories for each level. We found that a non-zero noise was necessary for most Atari games since deterministic policies learned through behavior cloning will often get stuck in a game and fail to take an action to continue

playing. For example, in Breakout, it is necessary to release the ball after it falls past the paddle, and a deterministic policy may fail to fire a new ball. For a similar reason, we also found it beneficial to include a few examples of no-op trajectories to encourage the agent to actually complete the game. For each game, we created an additional "no-op" demonstration set comprised of four length 500 no-op demonstrations. Without these no-op demonstrations, we found that often the learned reward function would give a small positive reward to the agent for just staying alive and sometimes the RL algorithm would decide to just sit at the start screen and accumulate a nearly indefinite stream of small rewards rather than play the game. Adding no-op demonstrations as the least preferred demonstrations shapes the reward function such that it encourages action and progress. While this does encode some amount of domain knowledge into the reward function, it is common that doing nothing is worse than actually attempting to complete a task. We note that in extremely risky scenarios, it may be the case that always taking the no-op action is optimal, but leave these types of domains for future work.

D.1.4 Noise Degradation

The full set of noise degradation plots for all seven Atari games are shown in Figure D.1. For MuJoCo experiments, we used 20 different noise levels evenly spaced over the interval $[0.0, 1.0)$ and generated 5 trajectories for each level. For the Atari experiments, we used the noise degradation schedule of $\mathcal{E} = (0.01, 0.25, 0.5, 0.75, 1.0)$ and generated $K = 20$ trajectories for each level.

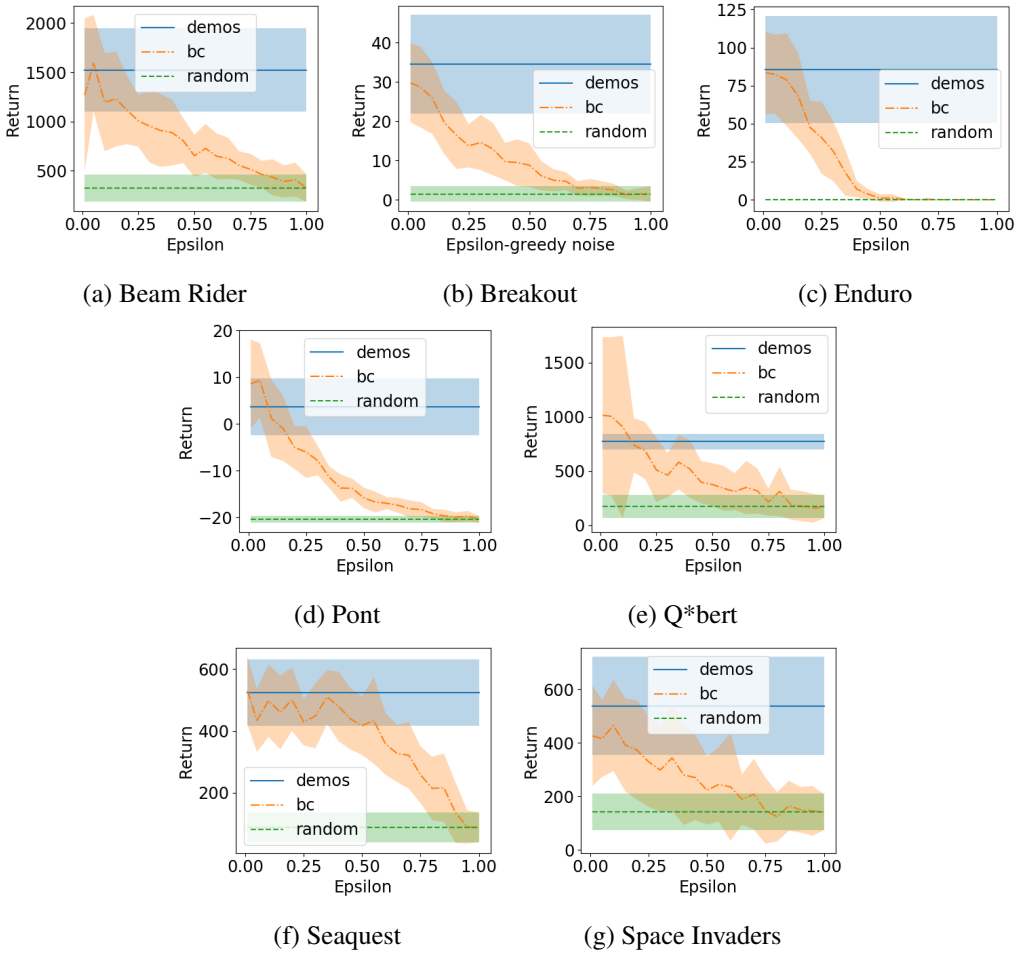
For the MuJoCo tasks, we used the following epsilon greedy policy:

$$\pi_{\mathrm{BC}}(s_t|\epsilon) = \begin{cases} \pi_{\mathrm{BC}}(s_t), & \text{with probability } 1-\epsilon \\ a_t \sim \mathcal{U}([-1,1]^n), & \text{with probability } \epsilon. \end{cases} \tag{D.1}$$

For Atari, we used the following epsilon greedy policy:

$$\pi_{\mathrm{BC}}(a_t|s_t,\epsilon) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|\mathcal{A}|}, & \text{if } \pi_{\mathrm{BC}}(s_t) = a_t \\ \frac{\epsilon}{|\mathcal{A}|}, & \text{otherwise} \end{cases} \tag{D.2}$$

where $|\mathcal{A}|$ is the number of valid discrete actions in each environment.



(a) Beam Rider (b) Breakout (c) Enduro

(d) Pont (e) Q*bert

(f) Seaquest (g) Space Invaders

Figure D.1: The performance degradation of an imitation policy learned via behavioral cloning as the probability of taking a random action increases. Behavioral cloning is done on 10 demonstrations. Plots show mean and standard deviations over 20 rollouts per noise level.

D.1.5 Reward function training

For reward function training, we generally followed the setup used in Brown et al. (2019b). We build a dataset of paired trajectory snippets with ranking, first by choosing two trajectories from given demonstrations and synthetic demonstrations, then by subsampling a snippet from each of trajectory.

MuJoCo experiments We built 3 datasets having different 5,000 pairs and trained a reward function for each of dataset using a neural network. Then, the ensemble of three neural network was used for reinforcement learning step. When two trajectories are selected from synthetic demonstration set to build a dataset, we discarded a pair whose epsilon difference is smaller than 0.3. This stabilizes a reward learning process by eliminating negative samples. Also, when subsampling from a whole trajectory, we limited the maximum length of snippet as 50 while there is no limitation on the minimum length. We then trained each neural network for 1,000 interactions with Adam optimizer with a learning rate of 1e-4 and minibatch size of 64. Weight decay regularization is also used with a coefficient of 0.01. A 3-layer MLP with 256 units in the middle is used to parameterize a reward function.

Atari experiments To generate training samples, we performed data augmentation to generate 40,000 training pairs. We first sampled two noise levels ϵ_i and ϵ_j, we then randomly sampled one trajectory from each noise level. Finally, we randomly cropped each trajectory keeping between 50 and 200 frames. Following the advice in Brown et al. (2019b), we also enforced a progress constraint such that the randomly cropped snippet from the trajectory with lower noise started at an observation timestep no earlier than the start of the snippet from the higher noise level. To speed up learning, we also only kept every 4th observation. Because observations are stacks of four frames this only removes redundant information from the trajectory. We assigned each trajectory pair a label indicating which trajectory had the lowest noise level.

Given our 40,000 labeled trajectory pairs, we optimized the reward function $\hat{R}_\theta$ using Adam with a learning rate of 1e-5. We held out 20% of the data as a validation set

and optimized the reward function on the training data using the validation data for early stopping. In particular, after every 1000 updates we fully calculated the validation error of the current model. We stopped training once the validation error failed to improve for 6 consecutive calculations of the validation error.

We used an architecture having four convolutional layers with sizes 7x7, 5x5, 3x3, and 3x3, with strides 3, 2, 1, and 1. The 7x7 convolutional layer used 32 filters and each subsequent convolutional layer used 16 filters and LeakyReLU non-linearities. We then used a fully connected layer with 64 hidden units and a single scalar output. We fed in stacks of 4 frames with pixel values normalized between 0 and 1 and masked reward-related information from the scene; the game score and number of lives, the sector number and number of enemy ships left on Beam Rider, the bottom half of the dashboard for Enduro to mask the position of the car in the race, the number of divers found and the oxygen meter for Seaquest, and the power level and inventory for Hero.

D.1.6 Policy optimization

We optimized a policy by training a PPO agent on the learned reward function. We used the default hyperparameters in OpenAI Baselines.[16] Due to the variability that results from function approximation when using PPO, we trained models using seeds 0, 1, and 2 and reported the best results among them.

MuJoCo experiments We trained an agent for 1 million steps, and gradient is estimated for every 4,096 simulation steps. As same as the original OpenAI implementation, we normalized a reward with running mean and standard deviation. Model ensemble of three neural network is done by averaging such normalized reward.

Atari experiments 9 parallel workers are used to collect trajectories for policy gradient estimation. To reduce reward scaling issues, we followed the procedure proposed by Brown et al. (2019b) and normalized predicted rewards by feeding the output of $\hat{R}_\theta(s)$

[16] https://github.com/openai/baselines

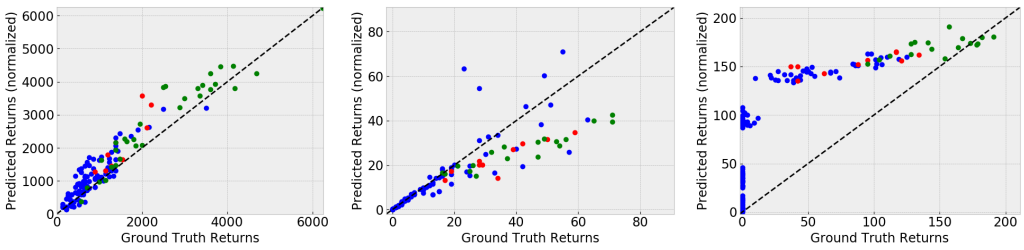
through a sigmoid function before passing it to PPO. We trained PPO on the learned reward function for 50 million frames to obtain our final policy.

D.2 GAIL

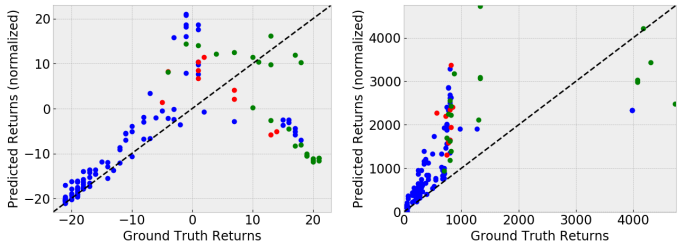
We used the default implementation of GAIL from OpenAI Baselines for Mujoco. For Atari we made a few changes to get the Baselines implementation to work with raw pixel observations. For the generator policy we used the Nature DQN architecture. The discriminator takes in a state (stack of four frames) and action (represented as a 2-d one-hot vector of shape (84,84,$|\mathcal{A}|$) that is concatenated to the 84x84x4 observation). The architecture for the discriminator is the same as the generator, except that it only outputs two logit values for discriminating between the demonstrations and the generator. We performed one generator update for every discriminator update.

D.3 D-REX Reward Extrapolation and Attention Heatmaps

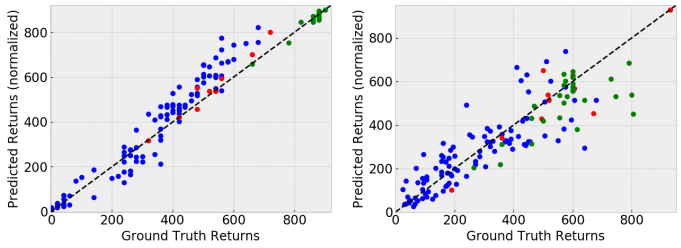
Figure D.2 shows the D-REX reward extrapolation plots for all seven Atari games. Figures D.3–D.9 show the D-REX reward heatmaps for all seven Atari games. We generated the heatmaps by taking a 3x3 mask and running it over every frame in an observation and compute the difference in predicted reward before and after the mask is applied. We then use the cumulative sum over all masks for each pixel to plot the heatmaps.



(a) Beam Rider (b) Breakout (c) Enduro



(d) Pong (e) Q*bert

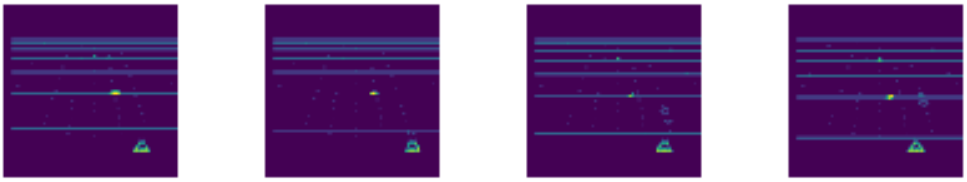


(f) Seaquest (g) Space Invaders

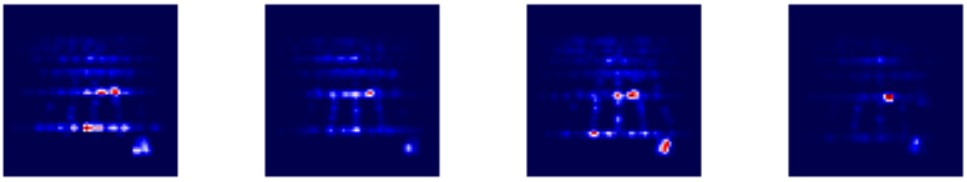
Figure D.2: Extrapolation plots for Atari games. Blue dots represent synthetic demonstrations generated via behavioral cloning with different amounts of noise injection. Red dots represent actual demonstrations, and green dots represent additional trajectories not seen during training. We compare ground truth returns over demonstrations to the predicted returns using D-REX (normalized to be in the same range as the ground truth returns).

(a) Beam Rider observation with maximum predicted reward using D-REX.

(b) Beam Rider reward model attention on maximum predicted reward using D-REX.



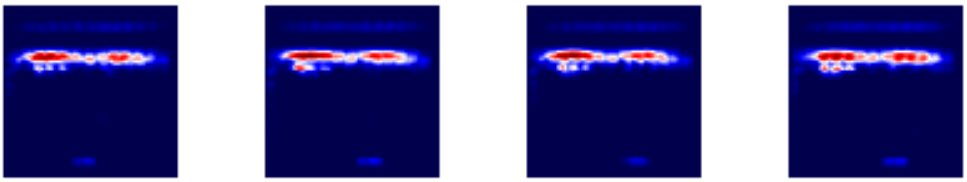
(c) Beam Rider observation with minimum predicted reward using D-REX.



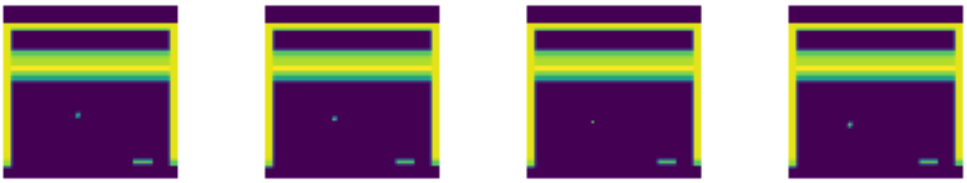
(d) Beam Rider reward model attention on minimum predicted reward using D-REX.

Figure D.3: D-REX maximum and minimum predicted observations and corresponding attention maps for Beam Rider across a held-out set of 15 demonstrations. The attention maps show that the reward is a function of the status of the controlled ship as well as the enemy ships and missiles.

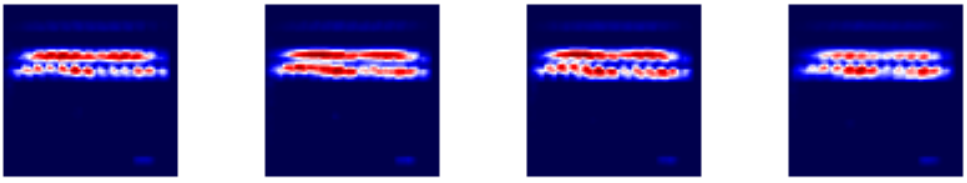
(a) Breakout observation with maximum predicted reward using D-REX.



(b) Breakout reward model attention on maximum predicted reward using D-REX.



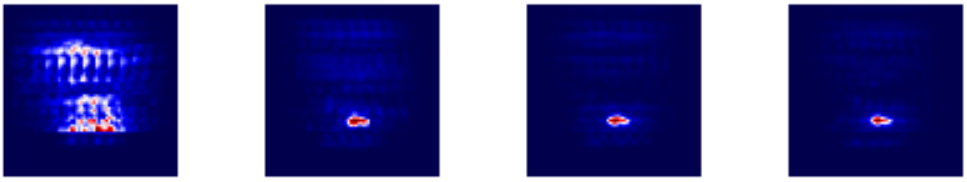
(c) Breakout observation with minimum predicted reward using D-REX.



(d) Breakout reward model attention on minimum predicted reward using D-REX.

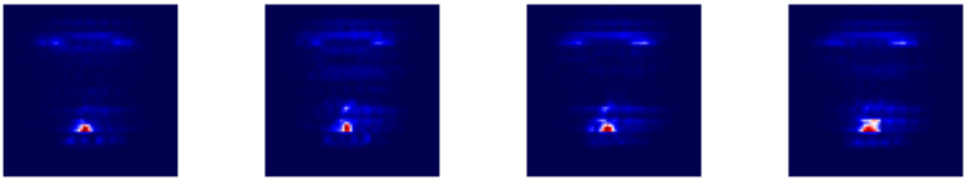
Figure D.4: D-REX maximum and minimum predicted observations and corresponding attention maps for Breakout across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows many of the bricks destroyed. The network has learned to put most of the reward weight on the remaining bricks. The observation with minimum predicted reward is an observation where none of the bricks have been destroyed.

(a) Enduro observation with maximum predicted reward using D-REX.



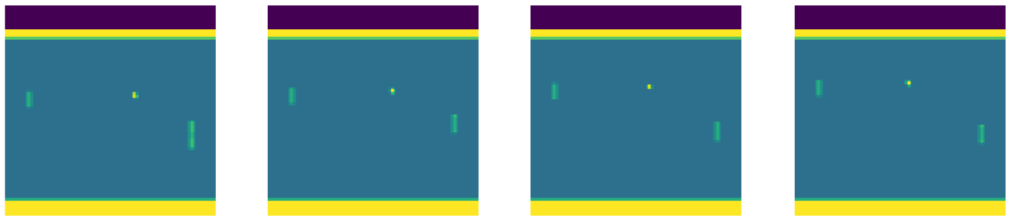
(b) Enduro reward model attention on maximum predicted reward using D-REX.

(c) Enduro observation with minimum predicted reward using D-REX.

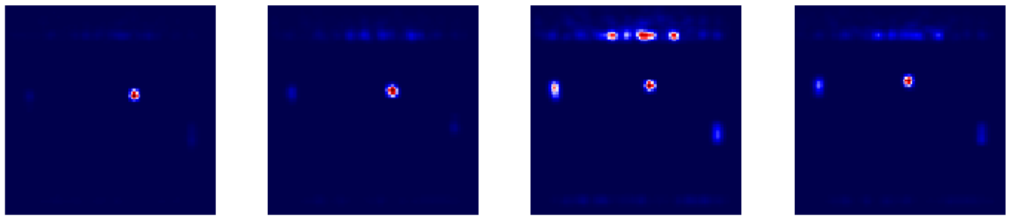


(d) Enduro reward model attention on minimum predicted reward using D-REX.

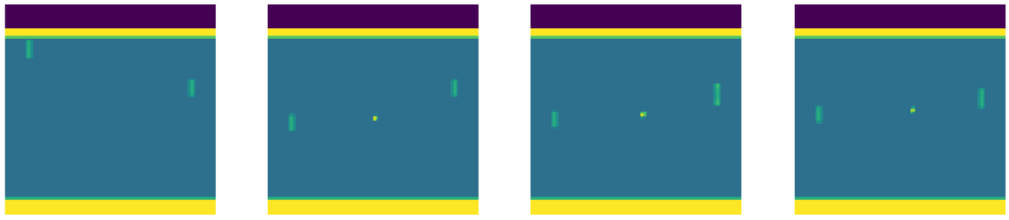
Figure D.5: D-REX maximum and minimum predicted observations and corresponding attention maps for Enduro across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows the car passing from one section of the race track to another as shown by the change in lighting. The observation with minimum predicted reward shows the controlled car falling behind another racer with attention focusing on the car being controlled as well as the speedometer.



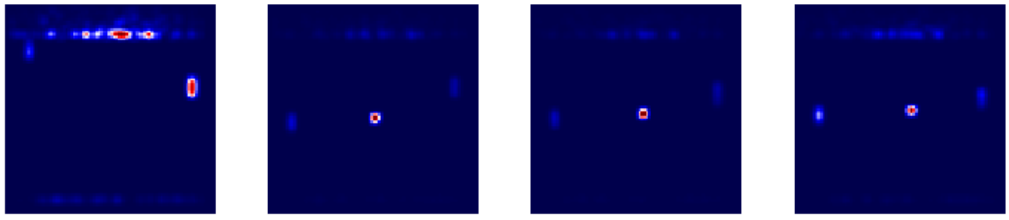
(a) Pong observation with maximum predicted reward using D-REX.



(b) Pong reward model attention on maximum predicted reward using D-REX.

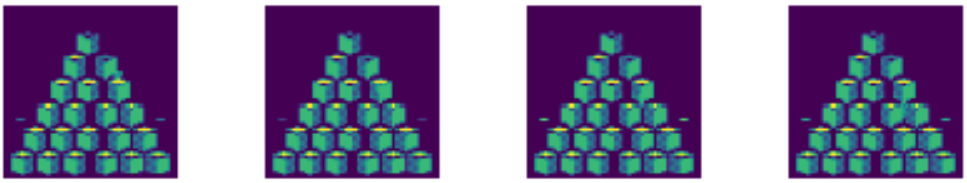


(c) Pong observation with minimum predicted reward using D-REX.

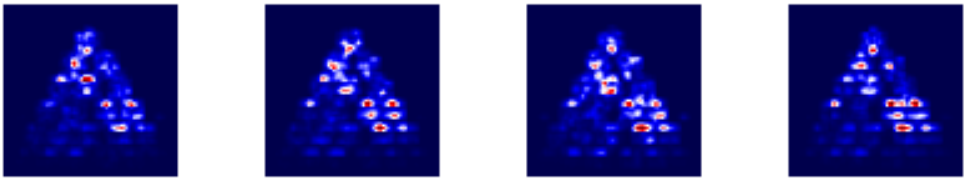


(d) Pong reward model attention on minimum predicted reward using D-REX.

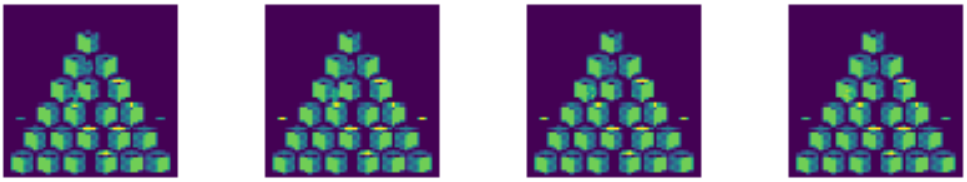
Figure D.6: D-REX maximum and minimum predicted observations and corresponding attention maps for Pong across a held-out set of 15 demonstrations. The network attends to the ball and paddles along with some artifacts outside the playing field. The observation with minimum predicted reward shows the ball being sent back into play after the opponent has scored.



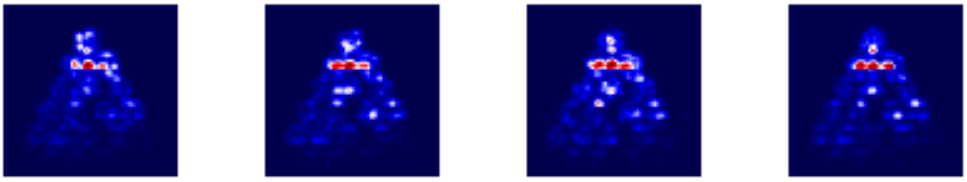
(a) Q*bert observation with maximum predicted reward using D-REX.



(b) Q*bert reward model attention on maximum predicted reward using D-REX.

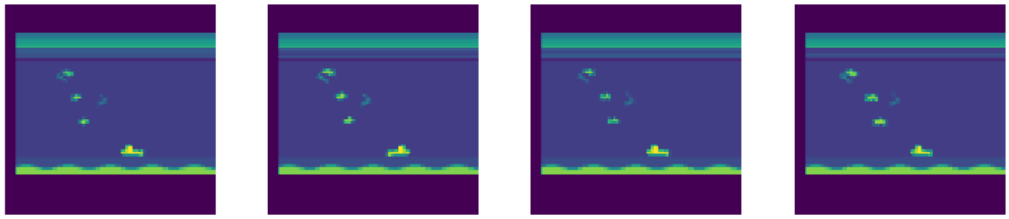


(c) Q*bert observation with minimum predicted reward using D-REX.

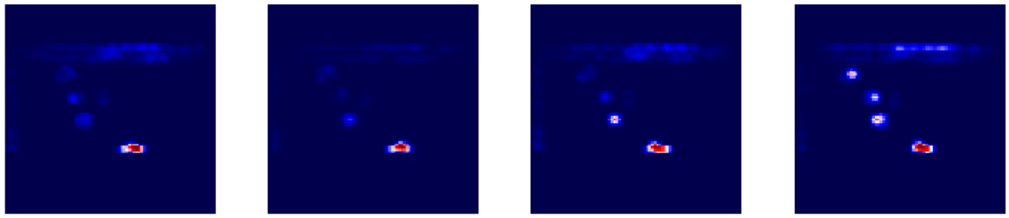


(d) Q*bert reward model attention on minimum predicted reward using D-REX.

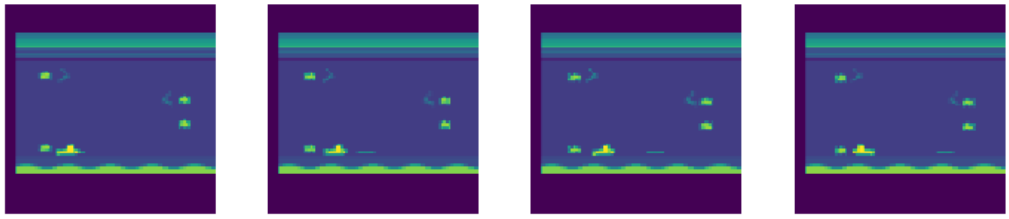
Figure D.7: D-REX maximum and minimum predicted observations and corresponding attention maps for Q*bert across a held-out set of 15 demonstrations. The network attention is focused on the different stairs, but is difficult to attribute any semantics to the attention maps.



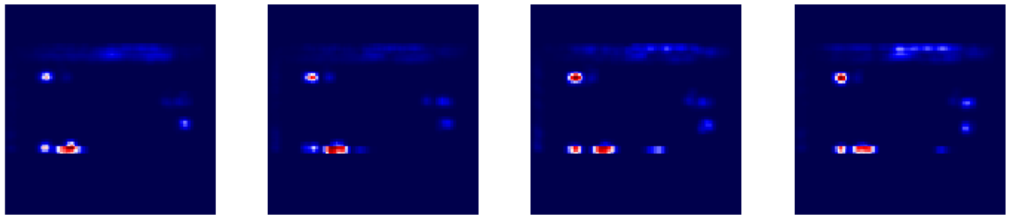
(a) Seaquest observation with maximum predicted reward using D-REX.



(b) Seaquest reward model attention on maximum predicted reward using D-REX.



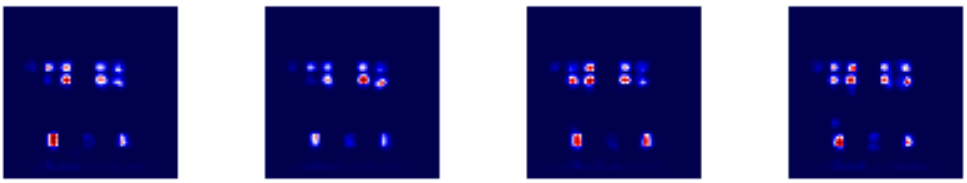
(c) Seaquest observation with minimum predicted reward using D-REX.



(d) Seaquest reward model attention on minimum predicted reward using D-REX.

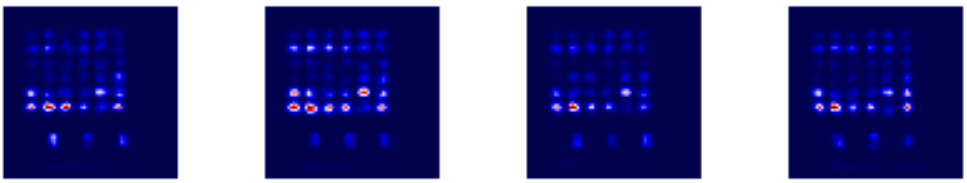
Figure D.8: D-REX maximum and minimum predicted observations and corresponding attention maps for Seaquest across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows the submarine in a safe location with no immediate threats. The observation with minimum predicted reward shows the submarine one frame before it is hit and destroyed by an enemy shark. This is an example of how the network has learned a shaped reward that helps it play the game better than the demonstrator. The network has learned to give most attention to nearby enemies and to the controlled submarine.

(a) Space Invaders observation with maximum predicted reward using D-REX.



(b) Space Invaders reward model attention on maximum predicted reward using D-REX.

(c) Space Invaders observation with minimum predicted reward using D-REX.



(d) Space Invaders reward model attention on minimum predicted reward using D-REX.

Figure D.9: D-REX maximum and minimum predicted observations and corresponding attention maps for Space Invaders across a held-out set of 15 demonstrations. The observation with maximum predicted reward shows an observation where most of the aliens have been successfully destroyed and the protective barriers are still intact. The attention map shows that the learned reward function is focused on the barriers and aliens, with less attention to the location of the controlled ship. The observation with minimum predicted reward shows the very start of a game with all aliens still alive. The network attends to the aliens and barriers, with higher weight on the aliens and the barrier closest to the space ship.

Appendix E

Bayesian REX Supplementary Materials

E.1 MCMC Details

We represent R_θ as a linear combination of pre-trained features:

$$R_\theta(\tau) = \sum_{s \in \tau} w^T \phi(s) = w^T \sum_{s \in \tau} \phi(s) = w^T \Phi_\tau. \tag{E.1}$$

We pre-compute and cache $\Phi_{\tau_i} = \sum_{s \in \tau_i} \phi(s)$ for $i = 1, \ldots, m$ and the likelihood becomes

$$P(\mathcal{P}, D \mid R_\theta) = \prod_{(i,j) \in \mathcal{P}} \frac{e^{\beta w^T \Phi_{\tau_j}}}{e^{\beta w^T \Phi_{\tau_j}} + e^{\beta w^T \Phi_{\tau_i}}}. \tag{E.2}$$

We enforce constraints on the weight vectors by normalizing the output of the weight vector proposal such that $\|w\|_2 = 1$ and use a Gaussian proposal function centered on w with standard deviation σ. Thus, given the current sample w_t, the proposal is defined as $w_{t+1} = \texttt{normalize}(\mathcal{N}(w_t, \sigma))$, in which `normalize` divides by the L2 norm of the sample to project back to the surface of the L2-unit ball.

For all experiments, except Seaquest, we used a default step size of 0.005. For Seaquest increased the step size to 0.05. We run 200,000 steps of MCMC and use a burn-in of 5000 and skip every 20th sample to reduce auto-correlation. We initialize the MCMC chain with a randomly chosen vector on the L2-unit ball. Because the inverse reinforcement learning is ill-posed there are an infinite number of reward functions that could match any set of demonstrations. Prior work by Finn et al. (2016) demonstrates that strong regularization is needed when learning cost functions via deep neural networks. To ensure that the rewards learned allow good policy optimization when fed into an RL algorithm we used a non-negative return prior on the return of the lowest ranked demonstration. The prior takes the following form:

$$\log P(w) = \begin{cases} 0 & \text{if } e^{\beta w^T \Phi_{\tau_1}} < 0 \\ -\infty & \text{otherwise} \end{cases} \tag{E.3}$$

This forces MCMC to not only find reward function weights that match the rankings, but to also find weights such that the return of the worse demonstration is non-negative. If the return of the worse demonstration was negative during proposal generation, then we assigned it a prior probability of $-\infty$. Because the ranking likelihood is invariant to affine transformations of the rewards, this prior simply shifts the range of learned returns and does not affect the log likelihood ratios.

E.2 Pre-training Latent Reward Features

We experimented with several pretraining methods. One method is to train R_θ using the pairwise ranking likelihood function in Equation (E.2) and then freeze all but the last layer of weights; however, the learned embedding may overfit to the limited number of preferences over demonstrations and fail to capture features relevant to the ground-truth reward function. Thus, we supplement the pairwise ranking objective with auxiliary objectives that can be optimized in a self-supervised fashion using data from the demonstrations.

Table E.1: Self-supervised learning objectives used to pre-train $\phi(s)$.

Inverse Dynamics	$f_{\text{ID}}(\phi(s_t), \phi(s_{t+1})) \rightarrow a_t$
Forward Dynamics	$f_{\text{FD}}(\phi(s_t), a_t) \rightarrow s_{t+1}$
Temporal Distance	$f_{\text{TD}}(\phi(s_t), \phi(s_{t+x}) \rightarrow x$
Variational Autoencoder	$f_A(\phi(s_t)) \rightarrow s_t$

We use the following self-supervised tasks to pre-train R_θ: (1) Learn an inverse dynamics model that uses embeddings $\phi(s_t)$ and $\phi(s_{t+1})$ to predict the corresponding action a_t Torabi et al. (2018a); Hanna and Stone (2017), (2) Learn a forward dynamics model that predicts s_{t+1} from $\phi(s_t)$ and a_t Oh et al. (2015); Thananjeyan et al. (2019), (3) Learn an embedding $\phi(s)$ that predicts the temporal distance between two randomly chosen states from the same demonstration Aytar et al. (2018), and (4) Train a variational pixel-to-pixel autoencoder in which $\phi(s)$ is the learned latent encoding Makhzani and Frey (2017); Doersch (2016). Table 6.1 summarizes the auxiliary tasks used to train $\phi(s)$.

There are many possibilities for pre-training $\phi(s)$; however, we found that each objective described above encourages the embedding to encode different features. For example, an accurate inverse dynamics model can be learned by only attending to the movement of the agent. Learning forward dynamics supplements this by requiring $\phi(s)$ to encode information about short-term changes to the environment. Learning to predict the temporal distance between states in a trajectory forces $\phi(s)$ to encode long-term progress. Finally, the autoencoder loss acts as a regularizer to the other losses as it seeks to embed all aspects of the state.

In the Atari domain, input to the network is given visually as grayscale frames resized to 84×84. To provide temporal information, four sequential frames are stacked one on top of another to create a *framestack* which provides a brief snapshot of activity. The network architecture takes a framestack, applies four convolutional layers following a similar architecture to Christiano et al. (2017) and Brown et al. (2019b), with leaky ReLU units as non-linearities following each convolution layer. The convolutions follow the following

structure:

#	Filter size	Image size	Stride
Input	-	$84 \times 84 \times 4$	-
1	$7x7$	$26 \times 26 \times 16$	3
2	$5x5$	$11 \times 11 \times 32$	2
3	$5x5$	$9 \times 9 \times 32$	1
4	$3x3$	$7 \times 7 \times 16$	1

The convolved image is then flattened. Two sequential fully connected layers, with leaky ReLU applied to the first layer, transform the flattened image into the encoding, $\phi(s)$ where s is the initial framestack. The width of these layers depends on the size of the feature encoding chosen. In our experiments with a latent dimension of 64, the first layer transforms from size 784 to 128 and the second from 128 to 64. See Figure 6.2 for a complete diagram of this process.

Architectural information for each auxiliary task is given below.

1. The variational autoencoder (VAE) tries to reconstruct the original framestack from the feature encoding using transposed convolutions. Mirroring the structure of the initial convolutions, two fully connected layers precede four transposed convolution layers. These first two layers transform the 64-dimensional feature encoding from 64 to 128, and from 128 to 1568. The following four layers' structures are summarized below:

#	Filter size	Image size	Stride
Input	-	$28 \times 28 \times 2$	-
1	$3x3$	$30 \times 30 \times 4$	1
2	$6x6$	$35 \times 35 \times 16$	1
3	$7x7$	$75 \times 75 \times 16$	2
4	$10x10$	$84 \times 84 \times 4$	1

A cross-entropy loss is applied between the reconstructed image and the original, as well as a term added to penalize the KL divergence of the distribution from the unit normal.

2. A temporal difference estimator, which takes two random feature encodings from the same demonstration and predicts the number of timesteps in between. It is a single fully-connected layer, transforming the concatenated feature encodings into a scalar time difference. A mean-squared error loss is applied between the real difference and predicted.

3. An inverse dynamics model, which takes two sequential feature encodings and predicts the action taken in between. It is again a single fully-connected layer, trained as a classification problem with a binary cross-entropy loss over the discrete action set.

4. A forward dynamics model, which takes a concatenated feature encoding and action and predicts the next feature encoding with a single fully-connected layer. This is repeated 5 times, which increases the difference between the initial and final encoding. It is trained using a mean-squared error between the predicted and real feature encoding.

5. A T-REX loss, which samples feature encodings from two different demonstrations and tries to predict which one of them has preference over the other. This is done with a single fully-connected layer that transforms an encoding into scalar reward, and is then trained as a classification problem with a binary cross-entropy loss. A 1 is assigned to the demonstration sample with higher preference and a 0 to the demonstration sample with lower preference.

In order to encourage a feature encoding that has information easily interpretable via linear combinations, the temporal difference, T-REX, inverse dynamics, and forward dynamics tasks consist of only a single layer atop the feature encoding space rather than multiple layers.

To compute the final loss on which to do the backwards pass, all of the losses described above are summed with weights determined empirically to balance out their values.

E.2.1 Training specifics

We used an NVIDIA TITAN V GPU for training the embedding. We used the same 12 demonstrations used for MCMC to train the self-supervised and ranking losses described above. We sample 60,000 trajectory snippets pairs from the demonstration pool, where each snippet is between 50 and 100 timesteps long. We use a learning rate of 0.001 and a weight decay of 0.001. We make a single pass through all of the training data using batch size of 1 resulting in 60,000 updates using the Adam Kingma and Ba (2014) optimizer. For Enduro prior work Brown et al. (2019b) showed that full trajectories resulted in better performance than subsampling trajectories. Thus, for Enduro we subsample 10,000 pairs of entire trajectories by randomly selecting a starting time between 0 and 5 steps after the initial state and then skipping every t frames where t is chosen uniformly from the range $[3, 7)$ and train with two passes through the training data. When performing subsampling for either snippets or full trajectories, we subsample pairs of trajectories such that one is from a worse ranked demonstration and one is from a better ranked demonstration following the procedure outlined in Brown et al. (2019b).

E.2.2 Visualizations of Learned Features

Viewable here[17] is a video containing an Enduro demonstration trajectory, its decoding with respect to the pre-trained autoencoder, and a plot of the dimensions in the latent encoding over time. Observe how changes in the demonstration, such as turning right or left or a shift, correspond to changes in the plots of the feature embedding. We noticed that certain features increase when the agent passes other cars while other features decrease when the agent gets passed by other cars. This is evidence that the pretraining has learned features

[17] https://www.youtube.com/watch?v=DMf8kNH9nVg

that are relevant to the ground truth reward which gives +1 every time the agent passes a car and -1 every time the agent gets passed.

Viewable here[18] is a similar visualization of the latent space for Space Invaders. Notice how it tends to focus on the movement of enemy ships, useful for game progress in things such as the temporal difference loss, but seems to ignore the player's ship despite its utility in inverse dynamics loss. Likely the information exists in the encoding but is not included in the output of the autoencoder.

Viewable here[19] is visualization of the latent space for Breakout. Observe that breaking a brick often results in a small spike in the latent encoding. Many dimensions, like the dark green curve which begins at the lowest value, seem to invert as game progress continues on, thus acting as a measure of how much time has passed.

E.3 Imitation Learning Ablations for Reward Function Feature Pre-Training

Table E.2 shows the results of pre-training reward features only using different losses. We experimented with using only the T-REX Ranking loss Brown et al. (2019b), only the self-supervised losses shown in Table 1 of the main paper, and using both the T-REX ranking loss plus the self-supervised loss function. We found that performance varried over the different pre-training schemes, but that using Ranking + Self-Supervised achieved high performance across all games, clearly outperforming only using self-supervised losses and achieving superior performance to only using the ranking loss on 3 out of 5 games.

[18] https://www.youtube.com/watch?v=2uN5uD17H6M
[19] https://www.youtube.com/watch?v=8zgbD1fZOH8

Table E.2: Comparison of different reward feature pre-training schemes. Ground-truth average returns for several Atari games when optimizing the mean and MAP rewards found using Bayesian REX. Each algorithm is given the same 12 demonstrations with ground-truth pairwise preferences. The average performance for each IRL algorithm is the average over 30 rollouts.

	Ranking Loss		Self-Supervised		Ranking + Self-Supervised	
Game	Mean	MAP	Mean	MAP	Mean	MAP
Beam Rider	3816.7	4275.7	180.4	143.7	**5870.3**	5504.7
Breakout	389.9	**409.5**	360.1	367.4	393.1	390.7
Enduro	472.7	479.3	0.0	0.0	135.0	**487.7**
Seaquest	675.3	670.7	674.0	683.3	606.0	**734.7**
Space Invaders	**1482.0**	1395.5	391.2	396.2	961.3	1118.8

E.4 Suboptimal Demonstration Details

We used the same suboptimal demonstrations used for the T-REX experiments in Chapter 5 (Brown et al., 2019b). These demonstrations were obtained by running PPO on the ground truth reward and checkpointing every 50 updates using OpenAI Baselines Dhariwal et al. (2017). Brown et al. (2019b) make the checkpoint files available, so to generate the demonstration data we used their saved checkpoints and followed the instructions in their released code to generate the data for our algorithm[20]. We gave Bayesian REX these demonstrations as well as ground-truth rankings using the game score; however, other than the rankings, Bayesian REX has no access to the true reward samples. Following the recommendations of Brown et al. (2019b), we mask the game score and other parts of the game that are directly indicative of the game score such as the number of enemy ships left, the number of lives left, the level number, etc. See Brown et al. (2019b) for full details.

[20]Code from Brown et al. (2019b) was downloaded from https://github.com/hiwonjoon/ICML2019-TREX

E.5 Reinforcement Learning Details

We used the OpenAI Baselines implementation of Proximal Policy Optimization (PPO) Schulman et al. (2017); Dhariwal et al. (2017). We used the default hyperparameters for all games and all experiments. We run RL for 50 million frames and then take the final checkpoint to perform evaluations. We adapted the OpenAI Baselines code so even though the RL agent receives a standard preprocessed observation, it only receives samples of the reward learned via Bayesian REX, rather than the ground-truth reward. T-REX Brown et al. (2019b) uses a sigmoid to normalize rewards before passing them to the RL algorithm; however, we obtained better performance for Bayesian REX by feeding the unnormalized predicted reward $R_\theta(s)$ into PPO for policy optimization. We follow the OpenAI baselines default preprocessing for the framestacks that are fed into the RL algorithm as observations. We also apply the default OpenAI baselines wrappers the environments. We run PPO with 9 workers on an NVIDIA TITAN V GPU.

E.6 High-Confidence Policy Performance Bounds

In this section we describe the details of the policy performance bounds.

E.6.1 Policy Evaluation Details

We estimated $\Phi_{\pi_{\text{eval}}}$ using C Monte Carlo rollouts for each evaluation policy. Thus, after generating C rollouts, $\tau_1, \ldots, \tau_C$ from π_{eval} the feature expectations are computed as

$$\Phi_{\pi_{\text{eval}}} = \frac{1}{C}\left[\sum_{i=1}^{C}\sum_{s\in\tau_i}\phi(s)\right]. \tag{E.4}$$

We used $C = 100$ for all experiments.

E.6.2 Evaluation Policies

We evaluated several different evaluation policies. To see if the learned reward function posterior can interpolate and extrapolate we created four different evaluation policies: A, B, C, and D. These policies were created by running RL via PPO on the ground truth reward for the different Atari games. We then checkpointed the policy and selected checkpoints that would result in different levels of performance. For all games except for Enduro these checkpoints correspond to 25, 325, 800, and 1450 update steps using OpenAI baselines. For Enduro, PPO performance was stuck at 0 return until much later in learning. To ensure diversity in the evaluation policies, we chose to use evaluation policies corresponding to 3125, 3425, 3900, and 4875 steps. We also evaluated each game with a No-Op policy. These policies are often adversarial for some games, such as Seaquest, Breakout, and Beam Rider, since they allow the agent to live for a very long time without actually playing the game—a potential way to hack the learned reward since most learned rewards for Atari will incentivize longer gameplay.

The results for Beam Rider and Breakout are shown in the main paper. For completeness, we have included the high-confidence policy evaluation results for the other games here in the Appendix. Table E.3 shows the high-confidence policy evaluation results for Enduro. Both the average returns over the posterior as well as the the high-confidence performance bounds ($\delta = 0.05$) demonstrate accurate predictions relative to the ground-truth performance. The No-Op policy results in the racecar slowly moving along the track and losing the race. This policy is accurately predicted as being much worse than the other evaluation policies. We also evaluated the Mean and MAP policies found by optimizing the Mean reward and MAP reward from the posterior obtained using Bayesian REX. We found that the learned posterior is able to capture that the MAP policy is more than twice as good as the evaluation policy D and that the Mean policy has performance somewhere between the performance of policies B and C. These results show that Bayesian REX has the potential to predict better-than-demonstrator performance Brown et al. (2019a).

Table E.3: Policy evaluation statistics for Enduro over the return distribution from the learned posterior $P(R|D,\mathcal{P})$ compared with the ground truth returns using game scores. Policies A-D correspond to checkpoints of an RL policy partially trained on the ground-truth reward function and correspond to 25, 325, 800, and 1450 training updates to PPO. No-Op that always plays the no-op action, resulting in high mean predicted performance but low 95%-confidence return (0.05-VaR).

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
A	324.7	48.2	7.3	3322.4
B	328.9	52.0	26.0	3322.4
C	424.5	135.8	145.0	3389.0
D	526.2	192.9	199.8	3888.2
Mean	1206.9	547.5	496.7	7249.4
MAP	395.2	113.3	133.6	3355.7
No-Op	245.9	-31.7	0.0	3322.0

Table E.4 shows the results for high-confidence policy evaluation for Seaquest. The results show that high-confidence performance bounds are able to accurately predict that evaluation policies A and B are worse than C and D. The ground truth performance of policies C and D are too close and the mean performance over the posterior and 0.05-VaR bound on the posterior are not able to find any statistical difference between them. Interestingly the no-op policy has very high mean and 95%-confidence lower bound, despite not scoring any points. However, as shown in the bottom half of Table E.4, adding one more ranked demonstration from a 3000 length segment of a no-op policy solves this problem. These results motivate a natural human-in-the-loop approach for safe imitation learning.

Finally, Table E.5 shows the results for high-confidence policy evaluation for Space Invaders. The results show that using both the mean performance and 95%-confidence lower bound are good indicators of ground truth performance for the evaluation polices. The No-Op policy for Space Invaders results in the agent getting hit by alien lasers early in the game. The learned reward function posterior correctly assigns low average performance and high risk (low 95%-confidence lower bound).

Table E.4: Policy evaluation statistics for Seaquest over the return distribution from the learned posterior $P(R|D, \mathcal{P})$ compared with the ground truth returns using game scores. Policies A-D correspond to checkpoints of an RL policy partially trained on the ground-truth reward function and correspond to 25, 325, 800, and 1450 training updates to PPO. No-Op always plays the no-op action, resulting in high mean predicted performance but low 0.05-quantile return (0.05-VaR). Results predict that No-Op is much better than it really is. However, simply adding a single ranked rollout from the No-Op policy and rerunning MCMC results in correct relative rankings with respect to the No-Op policy

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
A	24.3	10.8	338.6	1077.8
B	53.6	24.1	827.2	2214.1
C	56.0	25.4	872.2	2248.5
D	55.8	25.3	887.6	2264.5
No-Op	2471.6	842.5	0.0	99994.0
Results after adding one ranked demo from No-Op				
A	0.5	-0.5	338.6	1077.8
B	3.7	2.0	827.2	2214.1
C	3.8	2.1	872.2	2248.5
D	3.2	1.5	887.6	2264.5
No-Op	-321.7	-578.2	0.0	99994.0

Table E.5: Policy evaluation statistics for Space Invaders over the return distribution from the learned posterior $P(R|D, \mathcal{P})$ compared with the ground truth returns using game scores. Policies A-D correspond to checkpoints of an RL policy partially trained on the ground-truth reward function and correspond to 25, 325, 800, and 1450 training updates to PPO. The mean and MAP policies are the results of PPO using the mean and MAP rewards, respectively. No-Op that always plays the no-op action, resulting in high mean predicted performance but low 0.05-quantile return (0.05-VaR).

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
A	45.1	20.6	195.3	550.1
B	108.9	48.7	436.0	725.7
C	148.7	63.6	575.2	870.6
D	150.5	63.8	598.2	848.2
Mean	417.4	171.7	1143.7	1885.7
MAP	360.2	145.0	928.0	1629.5
NoOp	18.8	3.8	0.0	504.0

Appendix F

Supplementary Materials for Bayesian Robust Optimization for Imitation Learning

In this appendix we provide further implementation and hyperparameter details for the algorithms discussed in the main text.

F.1 Linear Programming Details

The BROIL objective is:

$$\begin{aligned} \underset{\boldsymbol{u}\in\mathbb{R}^{SA},\,\sigma\in\mathbb{R}}{\text{maximize}} \quad & (1-\lambda)\cdot\left(\sigma-\frac{1}{1-\alpha}\boldsymbol{p}^{\mathsf{T}}\left[\sigma\cdot\mathbf{1}-\boldsymbol{\Psi}(\pi,R)\right]_{+}\right)+\lambda\cdot(\boldsymbol{R}\boldsymbol{p})^{\mathsf{T}}\boldsymbol{u} \\ \text{subject to} \quad & \sum_{a\in\mathcal{A}}\left(\boldsymbol{I}-\gamma\cdot\boldsymbol{P}_a^{\mathsf{T}}\right)\boldsymbol{u}^a=\boldsymbol{p}_0, \quad \boldsymbol{u}\geq\mathbf{0}\,, \end{aligned}$$

This can be written as a linear program in standard form as follows:

$$\min_{\sigma,u} \quad -(1-\lambda)(\sigma + \frac{1}{1-\alpha}\boldsymbol{p}^\mathsf{T}\boldsymbol{z}) - \lambda \boldsymbol{p}^\mathsf{T}\boldsymbol{R}^\mathsf{T}u \tag{F.1}$$

$$\text{s.t.} \quad -\boldsymbol{R}^\mathsf{T}\boldsymbol{u} + \sigma\mathbf{1} - \boldsymbol{z} \leq -\boldsymbol{R}^\mathsf{T}\boldsymbol{u}_E \tag{F.2}$$

$$\left[(\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1}^\mathsf{T}), \ldots, (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_m}^\mathsf{T})\right] \begin{bmatrix} \boldsymbol{u}^{a_1} \\ \vdots \\ \boldsymbol{u}^{a_n} \end{bmatrix} = \boldsymbol{p}_0 \tag{F.3}$$

$$\boldsymbol{u} \geq \mathbf{0}, \boldsymbol{z} \geq \mathbf{0}, \sigma \in \mathbb{R} \tag{F.4}$$

We use Scipy's linear programming software (v 1.4.1) when solving this LP in the experiments in the paper [21].

F.2 Bayesian IRL Details

We use $\beta = 10$ for all of our experiments. We use a Gaussian proposal with standard deviation of 0.2. We use a burn-in period of 500 samples and skip every 5th sample after that to reduce autocorrelation. We tried a range of values for β and found very similar results. The step size was tuned to result in an accept ratio close to 0.4. Because scaling a reward function does not affect the optimal policy, we following prior work (Abbeel and Ng, 2004; Syed et al., 2008; Brown and Niekum, 2018) and assume that the reward function is scaled. For each proposal we project to the L2-norm ball to ensure that $\|w\|_2 = 1$.

F.3 Maximum Entropy IRL Detais

We compare against Maximum Entropy IRL (Ziebart et al., 2008). We use the implementation presented by Ziebart et al. (Ziebart et al., 2008), but to make it more comparable to

[21] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

Bayesian IRL we also add a boltzman parameter β to the likelihood such that

$$P(\xi) \propto \exp(\beta R(\xi)). \tag{F.5}$$

where ξ is a trajectory and $R(\xi)$ is the cumulative return of a trajectory. We used a horizon equal to the number of states in the MDP for the MaxEnt IRL algorithm Ziebart et al. (2008). We use $\beta = 10$ to match our implementation choice for Bayesian IRL. For gradient ascent, we used a learning rate of 0.01 and perform projected gradiant descent by projecting to the L2-norm ball such that $\|w\|_2 = 1$. We stop gradient ascent on the likelihood function once it has converged. Convergence is detected by measuring the difference in the L2-norm of the updated and prior weights and check if that is within a precision value of 0.00001. If so, then we stop gradient ascent. We experimented with several different values for each of these hyperparameters and found these to provide best performance.

F.4 LPAL Details

Linear Programming Apprenticeship Learning (LPAL) (Syed et al., 2008) has a robust form that is similar to ours but makes several critical and limiting assumptions: (1) they assume that the reward weights are strictly positive, this means they assume that the feature vector $\phi(s)$ explicitly encodes whether a feature is good or bad by its sign. (2) They assume very accurate estimation of the expert's expected feature counts $\hat{\mu}_E$. This requires an extremely large number of demonstrations (see Chapter 4). (3) Finally, they assume a worst-case adversarial reward function that will penalize whatever the learner does that is most different from the demonstrator, even if this reward function completeley contradicts the demonstrations, i.e., it does not take into account the likelihood of reward functions.

To compare BROIL against a state-of-the-art robust IRL approach, we implemented Linear Programming Apprenticeship Learning (Syed et al., 2008). The original paper assumes that the signs of the feature weights determine whether a feature is good or bad

and that the feature weights $\boldsymbol{w}$ lie on the probability simplex. In our work we do not assume prior knowledge about which features are good or bad (we seek to infer this from demonstrations). Thus, we implemented LPAL in a way that allows it to work with any features and feature weights that can be both positive and negative. We simply assume that $\|w\|_1 \leq 1$.

The original formulation of LPAL (which assumes knowledge of which reward features are good and bad) is as follows:

$$\max_{u,B} \left\{ B \;\mid\; B\mathbf{1} \leq \Phi_{\text{LPAL}}^T u - \hat{\mu}_E, \sum_{a \in \mathcal{A}} (\boldsymbol{I} - \gamma \cdot P_a^{\mathsf{T}}) u_a = p_0, u \geq \mathbf{0}, B \in \mathbb{R} \right\} . \tag{F.6}$$

In the paper we compare against the solution to the following derivation of the LPAL algorithm which does not assume the weights are non-negative, thus removing the need to know beforehand which features are good or bad. We solve for the LPAL solution using the following linear program in standard form:

$$\begin{aligned}
-\min_{B,u} \quad & B && \text{(F.7)}\\
\text{s.t.} \quad & B\mathbf{1} - \Phi^{\mathsf{T}} u \leq -\hat{\mu}_E && \text{(F.8)}\\
& -B\mathbf{1} + \Phi^{\mathsf{T}} u \leq \hat{\mu}_E && \text{(F.9)}\\
& \left[(I - \gamma P_{a_1}^{\mathsf{T}}), \ldots, (I - \gamma P_{a_m}^{\mathsf{T}}) \right] \begin{bmatrix} u_{a_1} \\ \vdots \\ u_{a_n} \end{bmatrix} = p_0 && \text{(F.10)}\\
& u \geq \mathbf{0} && \text{(F.11)}\\
& B \in \mathbb{R} && \text{(F.12)}
\end{aligned}$$

We derive this formulation of the maxmin objective for LPAL as follows. The basic

LPAL objective is

$$\max_{u \in U} \min_{w>=0, \|w\|_1 \leq 1} (u^t \Phi w - u_E \Phi w) \tag{F.13}$$

If we want to get rid of the requirement for positive weights then we have

$$\max_{u \in U} \min_{\|w\|_1 \leq 1} (u^t \Phi w - u_E \Phi w) \tag{F.14}$$

The inner minimization can be changed into a maximization as follows:

$$\max_{u \in U} - \max_{||w||_1 \leq 1} -((u^t \Phi - u_E \Phi) w) \tag{F.15}$$

Next we use the fact that the infinity norm and 1-norm are dual to each other and that $||z|| = || - z||$ to get the following optimization problem:

$$\max_{u \in U} -||u^T \Phi - u_E^T \Phi||_\infty \tag{F.16}$$

We change the minimization to a maximization by changing signs:

$$- \min_{u \in U} \|u^T \Phi - u_E^T \Phi\|_\infty \tag{F.17}$$

Using a standard linear programming trick we can write the above objective as follows:

$$- \min_{u \in U, B} \left\{ B \mid B\mathbf{1} \geq u^T \Phi - u_E^T \Phi, -B\mathbf{1} \geq -u^T \Phi + u_E^T \Phi, B \in \mathbb{R} \right\}. \tag{F.18}$$

Appendix G

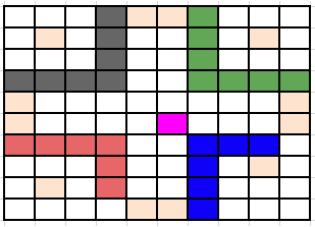
Supplementary Materials for Risk-Aware Active IRL

G.1 Comparing ActiveVaR and Random

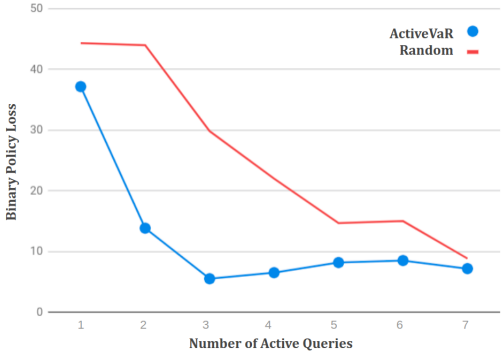
In the experiments of navigation in random gridworld (section 5.3), as shown in Figure 8.3, the improvement of ActiveVaR over Random is not prominent. We believe it is due to the fact that we used dense features and dense rewards that make random queries informative. Therefore, we ran additional experiments where we force the true reward and features to be sparse and the gridworld has only a few informative initial states such that there is a lower chance to sample a trajectory from an informative state.

Figure G.1 shows a selected setup and the policy loss averaged over 10 different runs in the selected environment. As shown in the plot, under this setting, ActiveVaR has a much larger improvement over random.

We also computed the worst-case actual policy loss, as this is what our method seeks to minimize, on a similar barrier domain with all possible states as initial states. In this setting random queries require on average 3.45 times more demonstrations than activeVaR queries to achieve low (< 0.01) worst-case policy loss.



(a) Gridworld Setup: each color (except light orange) is an unique feature; pink feature is the only feature with positive weight, other colors all have negative weights; states shaded with light orange are designated initial states.



(b) Binary Policy Loss Over Queries

Figure G.1: Gridworld navigation experiment with a sparse reward function consisting of a weighted combination of binary reward features.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st international conference on Machine learning*. xvii, 2, 10, 16, 22, 28, 30, 31, 35, 40, 42, 46, 49, 55, 98, 171, 224

Akgun, B., Cakmak, M., Yoo, J. W., and Thomaz, A. L. (2012). Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM. 53

Akrour, R., Schoenauer, M., and Sebag, M. (2011). Preference-based policy learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 12–27. Springer. 19, 92

Amin, K. and Singh, S. (2016). Towards resolving unidentifiability in inverse reinforcement learning. *arXiv preprint arXiv:1601.06569*. 13

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*. 2, 12, 13, 14, 55, 93, 108

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483. 11, 55

Armstrong, S. and Mindermann, S. (2018). Occam's razor is insufficient to infer the preferences of irrational agents. In *Advances in Neural Information Processing Systems*, pages 5598–5609. 53

Arora, S. and Doshi, P. (2018). A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877*. 11, 51

Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. (1999). Coherent measures of risk. *Mathematical finance*, 9(3):203–228. 15, 121

Asoh, H., Akaho, M. S. S., Kamishima, T., Hasida, K., Aramaki, E., and Kohro, T. (2013). An application of inverse reinforcement learning to medical records of diabetes treatment. In *ECML-PKDD Workshop on Reinforcement Learning with Generalized Feedback*. 123

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*. 99, 212

Babes, M., Marivate, V., Subramanian, K., and Littman, M. L. (2011). Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning*. 23

Bain, M. and Sommut, C. (1999). A framework for behavioural claning. *Machine intelligence*, 15(15):103. 80

Baker, C. L., Saxe, R., and Tenenbaum, J. B. (2009). Action understanding as inverse planning. *Cognition*, 113(3):329–349. 23

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065. 98

Barthe, F., Guédon, O., Mendelson, S., Naor, A., et al. (2005). A probabilistic approach to the geometry of the ℓ_p^n-ball. *The Annals of Probability*, 33(2):480–513. 159

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279. 20, 106

Ben-Tal, A., Bertsimas, D., and Brown, D. B. (2010). A Soft Robust Model for Optimization Under Ambiguity. *Operations Research*, 58(4):1220–1234. 15

Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009). *Robust Optimization*. Princeton University Press. 15

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer. 160

Biyik, E., Huynh, N., Kochenderfer, M. J., and Sadigh, D. (2020). Active preference-based gaussian process regression for reward learning. In *Proceedings of Robotics: Science and Systems (RSS)*. 152

Bıyık, E., Palan, M., Landolfi, N. C., Losey, D. P., and Sadigh, D. (2019). Asking easy questions: A user-friendly approach to active reward learning. In *Conference on Robot Learning (CoRL)*. 94, 151

Bobu, A., Bajcsy, A., Fisac, J. F., and Dragan, A. D. (2018). Learning under misspecified objective spaces. In *Conference on Robot Learning*, pages 796–805. 149, 154

Boularias, A., Kober, J., and Peters, J. (2011). Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189. 13

Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345. 59, 94

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*. 59, 61

Brown, D. S. (2011). Learning and control techniques in portfolio optimization. Undergraduate honors thesis, Brigham Young University. 117

Brown, D. S., Coleman, R., Srinivasan, R., and Niekum, S. (2020). Safe imitation learning via fast bayesian reward inference from preferences. In *Proceedings of the 37th International Conference on Machine Learning, ICML*. 2, 5, 92

Brown, D. S., Cui, Y., and Niekum, S. (2018). Risk-aware active inverse reinforcement learning. In *Proceedings of the 2nd Annual Conference on Robot Learning (CoRL)*. 2, 91, 134

Brown, D. S., Goo, W., and Niekum, S. (2019a). Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Proceedings of the 3rd Conference on Robot Learning*. 2, 5, 49, 94, 105, 219

Brown, D. S., Goo, W., Prabhat, N., and Niekum, S. (2019b). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *Proceedings of the 36th International Conference on Machine Learning, ICML*. xix, 2, 4, 11, 49, 75, 79, 80, 82, 88, 94, 106, 107, 196, 199, 200, 212, 215, 216, 217, 218

Brown, D. S. and Niekum, S. (2017). Toward probabilistic safety bounds for robot learning from demonstration. In *AAAI Fall Symposium on AI for HRI*. 4, 26

Brown, D. S. and Niekum, S. (2018). Efficient Probabilistic Performance Bounds for Inverse Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*. 2, 4, 24, 26, 103, 122, 171, 224

Brown, D. S. and Niekum, S. (2019a). Deep bayesian reward learning from preferences. In *NeurIPS Workshop on Safety and Robustness in Decision Making*. 5, 92

Brown, D. S. and Niekum, S. (2019b). Machine teaching for inverse reinforcement learning: Algorithms and applications. In *AAAI Conference on Artificial Intelligence*. 53, 169

Burchfiel, B., Tomasi, C., and Parr, R. (2016). Distance minimization for reward learning from scored trajectories. In *AAAI*, pages 3330–3336. 19

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. 81, 90

Castro, P. S., Li, S., and Zhang, D. (2019). Inverse reinforcement learning with multiple ranked experts. *arXiv preprint arXiv:1907.13411*. 52, 167

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*. 152

Chen, W., Liu, T.-Y., Lan, Y., Ma, Z.-M., and Li, H. (2009). Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323. 90

Choi, J. and Kim, K.-E. (2011). Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing Systems*. 24, 28, 35, 149

Choi, S., Lee, K., and Oh, S. (2019). Robust learning from demonstrations with mixed qualities using leveraged gaussian processes. *IEEE Transactions on Robotics*. 12

Chow, Y., Tamar, A., Mannor, S., and Pavone, M. (2015). Risk-sensitive and robust decision-making : A CVaR optimization approach. In *Neural Information Processing Systems (NIPS)*. 14, 117

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307. 19, 49, 54, 59, 88, 92, 94, 151, 183, 212

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765. 75

Chuck, C., Laskey, M., Krishnan, S., Joshi, R., Fox, R., and Goldberg, K. (2017). Statistical data cleaning for deep learning of automation tasks from demonstrations. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1142–1149. IEEE. 53

Cohn, R., Durfee, E., and Singh, S. (2011). Comparing action-query strategies in semi-autonomous agents. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*. 18, 35, 37, 42, 134, 140

Cui, Y. and Niekum, S. (2018). Active reward learning from critiques. In *IEEE International Conference on Robotics and Automation (ICRA)*. xix, 18, 104, 105, 134, 137, 138, 140, 141, 151

de Haan, P., Jayaraman, D., and Levine, S. (2019). Causal confusion in imitation learning. In *Advances in Neural Information Processing Systems*, pages 11693–11704. 108

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. 75

Delage, E. and Mannor, S. (2010). Percentile optimization for markov decision processes with parameter uncertainty. *Operations research*, 58(1):203–213. 15, 117, 121, 125

Delbaen, F. (2002). Coherent risk measures on general probability spaces. In *Advances in finance and stochastics*, pages 1–37. Springer. 121

Derman, E., Mankowitz, D., Mann, T. A., and Mannor, S. (2018). Soft-Robust Actor-Critic Policy-Gradient. In *Uncertainty in Artificial Intelligence (UAI)*. 15

Desai, V. V., Farias, V. F., and Moallemi, C. C. (2012). Approximate dynamic programming via a smoothed linear program. *Operations Research*, 60(3):655–674. 152

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. https://github.com/openai/baselines. 60, 82, 98, 150, 217, 218

Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*. 99, 212

El Asri, L., Piot, B., Geist, M., Laroche, R., and Pietquin, O. (2016). Score-based inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 457–465. International Foundation for Autonomous Agents and Multiagent Systems. 19

Eric, B., Freitas, N. D., and Ghosh, A. (2008). Active preference learning with discrete choice data. In *Advances in neural information processing systems*, pages 409–416. 19

Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*. 10, 49, 55, 211

Fisac, J. F., Akametalu, A. K., Zeilinger, M. N., Kaynama, S., Gillula, J., and Tomlin, C. J. (2018). A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*. 2

Föllmer, H. and Knispel, T. (2011). Entropic risk measures: Coherence vs. convexity, model ambiguity and robust large deviations. *Stochastics and Dynamics*, 11(02n03):333–351. 148

Follmer, H. and Schied, A. (2011). *Stochastic Finance: An Introduction in Discrete Time*. Walter de Gruyter, 3rd edition. 15

Fu, J., Luo, K., and Levine, S. (2017). Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*. 2, 49

Gao, Y., Lin, J., Yu, F., Levine, S., Darrell, T., et al. (2018). Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*. 12

Gao, Y., Peters, J., Tsourdos, A., Zhifei, S., and Meng Joo, E. (2012). A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311. 11

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480. 2, 3, 13, 14, 117

Ghavamzadeh, M., Petrik, M., and Chow, Y. (2016). Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306. 117

Goo, W. and Niekum, S. (2019). One-shot learning of multi-step tasks from observation via activity localization in auxiliary video. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*. 11

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680. 10, 11

Greydanus, S., Koul, A., Dodge, J., and Fern, A. (2018). Visualizing and understanding atari agents. In *International Conference on Machine Learning*, pages 1787–1796. 84, 184

Grollman, D. H. and Billard, A. (2011). Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3804–3809. IEEE. 12

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. (2017). Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774. 15, 118, 120, 122, 149, 153

Hanna, J. P. and Stone, P. (2017). Grounded action transformation for robot learning in simulation. In *Thirty-First AAAI Conference on Artificial Intelligence*. 99, 212

Hanna, J. P., Stone, P., and Niekum, S. (2017). Bootstrapping with models: Confidence intervals for off-policy evaluation. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*. 14

Henderson, P., Chang, W.-D., Bacon, P.-L., Meger, D., Pineau, J., and Precup, D. (2018). Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 11, 55

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 93

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 12, 49, 88, 183

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573. xvii, xviii, xix, xxiv, 10, 11, 15, 16, 49, 61, 62, 64, 65, 85, 86, 106, 107

Huang, J., Wu, F., Precup, D., and Cai, Y. (2018). Learning safe policies with expert guidance. In *Advances in Neural Information Processing Systems*, pages 9105–9114. 15, 16, 118

Iancu, D. A. and Trichakis, N. (2014). Pareto Efficiency in Robust Optimization. *Management Science*, 60(1):130–147. 15

Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., and Amodei, D. (2018). Reward learning from human preferences and demonstrations in atari. In *Advances in Neural Information Processing Systems*. xxi, 19, 59, 63, 65, 79, 93, 108, 151, 183, 186

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373. 1

Jacq, A., Geist, M., Paiva, A., and Pietquin, O. (2019). Learning from a learner. In *International Conference on Machine Learning*, pages 2990–2999. 12, 75, 94

Jorion, P. (1997). *Value at risk*. McGraw-Hill, New York. 14, 31, 32, 102, 117, 135

Kalakrishnan, M., Pastor, P., Righetti, L., and Schaal, S. (2013). Learning objective functions for manipulation. In *2013 IEEE International Conference on Robotics and Automation*, pages 1331–1336. IEEE. 13

Kalantari, J., Nelson, H., and Chia, N. (2020). The unreasonable effectiveness of inverse reinforcement learning in advancing cancer research. In *Association for the Advancement of Artificial Intelligence (AAAI)*. 123

Karasev, V., Ayvaci, A., Heisele, B., and Soatto, S. (2016). Intent-aware long-term prediction of pedestrian motion. In *IEEE International Conference on Robotics and Automation*, pages 2543–2549. 23

Kent, D., Saldanha, C., and Chernova, S. (2017). A comparison of remote robot teleoperation interfaces for general object manipulation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 371–379. ACM. 53

Kim, B. and Pineau, J. (2016). Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics*, 8(1):51–66. 23

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 60, 215

Knox, W. B. and Stone, P. (2009). Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM. 17, 151

Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856. 12

Kurin, V., Nowozin, S., Hofmann, K., Beyer, L., and Leibe, B. (2017). The atari grand challenge dataset. *arXiv preprint arXiv:1705.10998*. xvii, 67, 70, 184

Lacotte, J., Ghavamzadeh, M., Chow, Y., and Pavone, M. (2019). Risk-sensitive generative adversarial imitation learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2154–2163. 15, 16, 117, 125

Laskey, M., Lee, J., Fox, R., Dragan, A., and Goldberg, K. (2017). Dart: Noise injection for robust imitation learning. *Conference on Robot Learning (CoRL)*. 13

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. (2017). Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*. 93

Levine, S., Popovic, Z., and Koltun, V. (2011). Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*. 23, 28

Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. 97

Liu, Y., Gupta, A., Abbeel, P., and Levine, S. (2018). Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE. 11

Lopes, M., Melo, F., and Montesano, L. (2009). Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. xxix, 18, 37, 134, 138, 139, 140

Lopes, M., Melo, F. S., and Montesano, L. (2007). Affordance-based imitation learning in robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1015–1021. 23

Luce, R. D. (2012). *Individual choice behavior: A theoretical analysis*. Courier Corporation. 59, 81

MacGlashan, J., Ho, M. K., Loftin, R., Peng, B., Wang, G., Roberts, D. L., Taylor, M. E., and Littman, M. L. (2017). Interactive learning from policy-dependent human feedback. In *International Conference on Machine Learning*, pages 2285–2294. 17

Majumdar, A., Singh, S., Mandlekar, A., and Pavone, M. (2017). Risk-sensitive inverse reinforcement learning via coherent risk models. In *Robotics: Science and Systems*. 16, 117

Makhzani, A. and Frey, B. J. (2017). Pixelgan autoencoders. In *Advances in Neural Information Processing Systems*, pages 1975–1985. 99, 212

Markowitz, H. M. and Todd, G. P. (2000). *Mean-variance analysis in portfolio choice and capital markets*, volume 66. John Wiley & Sons. 117

Michini, B. and How, J. P. (2012a). Bayesian nonparametric inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 23

Michini, B. and How, J. P. (2012b). Improving the efficiency of bayesian inverse reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pages 3651–3656. 38

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529. 93, 181

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287. 12, 55, 93, 169

Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 663–670. 2, 10, 15, 22, 53, 54, 116, 135

Ogryczak, W. and Ruszczyński, A. (1999). From stochastic dominance to mean-risk models: Semideviations as risk measures. *European Journal of Operational Research*, 116(1):33–50. 148

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871. 99, 212

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179. 11

Palan, M., Landolfi, N. C., Shevchuk, G., and Sadigh, D. (2019). Learning reward functions by integrating human demonstrations and preferences. In *Proceedings of Robotics: Science and Systems (RSS)*. 19, 79, 88, 92, 94

Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. (2013). Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624. 1

Pazis, J. and Parr, R. (2011). Non-parametric Approximate Linear Programming for MDPs. In *Conference on Artificial Intelligence (AAAI)*. 152

Petrik, M. (2010). *Optimization-based Approximate Dynamic Programming*. PhD thesis, University of Massachusetts Amherst. 152

Petrik, M. and Russell, R. H. (2019). Beyond confidence regions: Tight bayesian ambiguity sets for robust mdps. *arXiv preprint arXiv:1902.07605*. 14

Petrik, M. and Subramanian, D. (2012). An approximate solution method for large risk-averse Markov decision processes. In *Uncertainty in Artificial Intelligence (UAI)*. 117

Pirotta, M. and Restelli, M. (2016). Inverse reinforcement learning through policy gradient minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1993–1999. 22, 29

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97. 1, 9

Pradier, M. F., Pan, W., Yao, J., Ghosh, S., and Doshi-Velez, F. (2018). Projected bnns: Avoiding weight-space pathologies by learning latent representations of neural network weights. *arXiv preprint arXiv:1811.07006*. 96

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons. 8, 21, 119, 122, 123

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artifical intelligence*, pages 2586–2591. 4, 10, 23, 24, 28, 33, 35, 49, 55, 91, 93, 102, 118, 120, 122, 128, 132, 136

Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42. 14, 117, 118, 121, 122, 148

Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. 177, 179

Ross, S., Gordon, G., and Bagnell, D. (2011). 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*, pages 627–635. 9, 17

Rothkopf, C. A. and Ballard, D. H. (2013). Modular inverse reinforcement learning for visuomotor behavior. *Biological cybernetics*, 107(4):477–490. 23

Russell, R. H. and Petrik, M. (2019). Beyond Confidence Regions: Tight Bayesian Ambiguity Sets for Robust MDPs. In *Advances in Neural Information Processing Systems (NeurIPS)*. 15

Russell, S. and Norvig, P. (2002). Artificial intelligence: a modern approach. 118

Sadigh, D., Dragan, A. D., Sastry, S. S., and Seshia, S. A. (2017). Active preference-based learning of reward functions. In *Proceedings of Robotics: Science and Systems (RSS)*. 18, 19, 22, 28, 54, 79, 88, 92, 122, 123

Santara, A., Naik, A., Ravindran, B., Das, D., Mudigere, D., Avancha, S., and Kaul, B. (2017). Rail: Risk-averse imitation learning. *arXiv preprint arXiv:1707.06658*. 16

Sarafian, E., Tamar, A., and Kraus, S. (2018). Safe policy learning from observations. *arXiv preprint arXiv:1805.07805*. 12

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 60, 81, 106, 218

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. (2018). Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE. 11

Shah, R., Gundotra, N., Abbeel, P., and Dragan, A. (2019). On the feasibility of learning, rather than assuming, human biases for reward inference. In *International Conference on Machine Learning*, pages 5670–5679. 23

Shapiro, A., Dentcheva, D., and Ruszczynski, A. (2014). *Lectures on stochastic programming: Modeling and theory*. 15

Shiarlis, K., Messias, J., and Whiteson, S. (2016). Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. 12

Srinivas, A., Laskin, M., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*. 152

Sugiyama, H., Meguro, T., and Minami, Y. (2012). Preference-learning based inverse reinforcement learning for dialog control. In *INTERSPEECH*, pages 222–225. 19

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge. 2, 8, 10, 21, 33, 52, 74

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063. 74, 97

Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039. xxviii, 15, 16, 118, 122, 128, 129, 130, 131, 132, 224, 225

Syed, U. and Schapire, R. E. (2007). A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456. xvii, 12, 16, 29, 31, 35, 41, 42, 46, 156

Tamar, A., Glassner, Y., and Mannor, S. (2015). Optimizing the cvar via sampling. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2993–2999. 14, 31, 117

Tang, Y. C., Zhang, J., and Salakhutdinov, R. (2020). Worst cases policy gradients. In Kaelbling, L. P., Kragic, D., and Sugiura, K., editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1078–1093. PMLR. 14, 117

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685. 93

Taylor, M. E., Suay, H. B., and Chernova, S. (2011). Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems. 12

Thananjeyan, B., Balakrishna, A., Rosolia, U., Li, F., McAllister, R., Gonzalez, J. E., Levine, S., Borrelli, F., and Goldberg, K. (2019). Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. *arXiv preprint arXiv:1905.13402*. 1, 99, 212

Thomas, P., Theocharous, G., and Ghavamzadeh, M. (2015a). High confidence policy improvement. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2380–2388. 14

Thomas, P. S., da Silva, B. C., Barto, A. G., Giguere, S., Brun, Y., and Brunskill, E. (2019).

Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004. 2, 13

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. (2015b). High-confidence off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3000–3006. 14, 27

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE. 59

Torabi, F., Warnell, G., and Stone, P. (2018a). Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. xvii, xxiv, 1, 9, 11, 55, 61, 62, 64, 65, 99, 149, 151, 167, 181, 212

Torabi, F., Warnell, G., and Stone, P. (2018b). Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*. 11

Tucker, A., Gleave, A., and Russell, S. (2018). Inverse reinforcement learning for video games. In *Proceedings of the Workshop on Deep Reinforcement Learning at NeurIPS*. 11, 65

Volkovs, M. N. and Zemel, R. S. (2014). New learning methods for supervised and unsupervised preference aggregation. *The Journal of Machine Learning Research*, 15(1):1135–1176. 94

Weisstein, E. (2017). Ball point picking. *From MathWorld–A Wolfram Web Resource. http://mathworld. wolfram. com/BallPointPicking. html*. 159

Wilson, A., Fern, A., and Tadepalli, P. (2012). A bayesian approach for policy learning from trajectory preference queries. In *Advances in neural information processing systems*, pages 1133–1141. 92

Wirth, C., Akrour, R., Neumann, G., and Fürnkranz, J. (2017). A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46. 19

Wirth, C., Fürnkranz, J., and Neumann, G. (2016). Model-free preference-based reinforcement learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2222–2228. AAAI Press. 19

Wulfmeier, M., Ondruska, P., and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*. 28

Wulfmeier, M., Rao, D., Wang, D. Z., Ondruska, P., and Posner, I. (2017). Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research*, 36(10):1073–1087. 123

Wyrobek, K. A., Berger, E. H., Van der Loos, H. M., and Salisbury, J. K. (2008). Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *2008 IEEE International Conference on Robotics and Automation*, pages 2165–2170. IEEE. 2

Xu, K., Ratner, E., Dragan, A., Levine, S., and Finn, C. (2019). Learning a prior over intent via meta-inverse reinforcement learning. *International Conference on Machine Learning*. 120, 125

Yu, T., Finn, C., Xie, A., Dasari, S., Zhang, T., Abbeel, P., and Levine, S. (2018). One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*. 11

Zhang, J. and Cho, K. (2017). Query-efficient imitation learning for end-to-end simulated driving. In *Thirty-First AAAI Conference on Artificial Intelligence*. 15

Zheng, J., Liu, S., and Ni, L. M. (2014). Robust bayesian inverse reinforcement learning

with sparse behavior noise. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2198–2205. 12, 16, 149

Zhu, Y., Wang, Z., Merel, J., Rusu, A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., et al. (2018). Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*. 1

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 1433–1438. xxviii, 10, 13, 15, 22, 28, 49, 55, 118, 128, 130, 131, 132, 149, 224, 225