Combining Deep Bayesian Inverse Reinforcement Learning from Preferences (B-REX) with Bayesian Robust Optimization for Imitation Learning (BROIL)

Austin Nguyen, Jerry Zhu, Peter Zhu, Daniel Brown*

University of California, Berkeley

December 2020

Abstract

One of the main challenges in Bayesian inverse reinforcement learning (IRL) methods is reasoning about reward uncertainty and determining policy safety when learning from generated reward functions. In addition, most Bayesian IRL methods are computationally intractable for high dimensional problems due to the requirement of a Markov Decision Process (MDP) when sampling from posterior reward distributions. Bayesian Reward Extrapolation (B-REX) helps solve this issue by using a preference-based Bayesian reward learning algorithm that is easily scalable to high dimensional problems. B-REX uses successor feature representations and preferences over multiple demonstrations to generate samples from a posterior reward distribution without needing an MDP solver.

However, training policies naively over B-REX's outputted distribution is dangerous, especially when determining what action an agent should take when outside the explored state distribution. Some existing safe imitation learning approaches based on IRL deal with this uncertainty by assuming an adversarial reward function. Fully adversarial reward functions lead to overly conservative, unfavorable policies.

We propose a method that connects B-REX to Bayesian Robust Optimization for Imitation Learning (BROIL), an algorithm that leverages Bayesian reward function inference as well as a user specific risk tolerance to efficiently optimize a robust policy that helps balance expected return and worst-case performance. In general, B-REX receives ranked demonstrations to develop a sub-optimal reward distribution. Then, BROIL receives B-REX's reward distribution to train an agent policy. Overall, the policy is able to balance return maximization and risk minimization from inputted ranked demonstrations without knowing the ground-truth reward.

The main focus of the research is on BROIL before integrating B-REX. BROIL is evaluated on 3 different environments each with hand-coded sub-optimal reward distributions: PointBot[1], Cheetah, and Ant. Preliminary results are demonstrated with CartPole as well. Once BROIL is combined with different reinforcement learning policy algorithms, we analyze performance through visualizations and training plots for each environment. As a result, we found that BROIL is adaptable to continuous state and action spaces, robust to sub-optimal reward functions, and comprehensive to complex environments. Incorporated algorithms include Vanilla Policy Gradient (VPG), Proximal Policy Optimization (PPO), and Reward-To-Go (RTG) from the Spinning Up repository on Open AI. We used B-REX to create our reward distribution for BROIL to learn a risk-tolerant policy and also designed custom reward distributions to evaluate BROIL's risk-aversion properties.

*Project Advisor

[1]Custom Environment from AUTOLab

1 Introduction

The primary goal of this research is to devise a risk-tolerant algorithm that is adaptable to complex environments and explore the creation of an end-to-end framework for learning from demonstrations under uncertainty. In real life applications of deep reinforcement learning, problems often involve reasoning of unstructured situations and safety of particular policies. After an extensive literature review of B-REX and BROIL, our team explored a novel algorithm that combines the two approaches. In addition, experiments on the adaptability of BROIL were conducted on varying environments, reward functions, and state and action spaces. B-REX learns its own reward function through a pairwise ranking likelihood of demonstrations (human and/or noise injection) to create a posterior distribution over reward functions. BROIL, on the other hand, utilizes a Bayesian reward function with a user specific tolerance to optimize a policy that balances expected return with conditional value of risk (CVaR). Together with these two modules, we generate pairwise preferences over demonstration trajectories, use B-REX to create a learned reward function, and use BROIL to optimize our policy, balancing expected return with CVaR.

2 Related Work

There has been work related to solving reinforcement learning (RL) problems with uncertain reward functions, reducing RL problems to lower dimensional spaces, and balancing risk tolerance to efficiently optimize a robust policy. In particular, the BROIL and B-REX algorithms have both been developed and tested for their specific use cases. However, many of these ideas have not been explored in combination, and comprehensive testing for adaptability on different environments has not been done before. Our research builds upon the referenced papers describing B-REX[3] and BROIL[4]. In particular, the focus will be on PointBot, Cheetah, and Ant which are new areas of application for these algorithms.

3 Bayesian Robust Optimization for Imitation Learning (BROIL)

3.1 Background

We model the BROIL environment as a Markov Decision Process (MDP) using tuples ($\mathcal{S}$, $\mathcal{A}$, r, P, γ, p_0). $\mathcal{S}$ =$\{s_1, \dots, S\}$ denotes the state space, $\mathcal{A} = \{1, \dots, A\}$ denotes the action space, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ corresponds to the reward function, P: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ describes the transition dynamics, $\gamma \in [0, 1)$ is the discount factor, $p_0 \in \Delta^S$ is the starting state distribution, and Δ^k is the probability simplex within k-dimensions.

$\pi_\theta : \mathcal{S} \times \mathcal{A} \to [0, 1]$ denotes a policy parameterized by θ, while $\pi_E : \mathcal{S} \times \mathcal{A}$ denotes the expert's policy when learning from demonstrations. The reward at each state by a policy is $r_\pi(s) = E_{a\sim\pi(s)}[r(s, a)]$, and the transition probability for policy π is $P_\pi(s, s') = E_{a\sim\pi(s)}[P(s, a, s')] = \sum_a \pi(a|s)P(s, a, s')$. We represent state-action occupancies of policy π as $u_\pi \in R^{S\times A}$. We denote the discounted state-action occupanies as $u^a_\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t 1_{s_t=s \wedge a_t=a}]$. As a result, we concatenate these results to get $u_\pi = ((u^{a_1}_\pi)^T, \dots, (u^{a_A}_\pi)^T)^T$. Therefore, the expected return of policy π is denoted by $\rho(\pi, r) = u^T_\pi r$ for reward function $r \in R^{S\times A}$.

3.2 Constructing Reward Functions

We assume that we do not have direct access to reward function due to uncertainty over the true reward function. We denote R as the true reward function and the posterior distribution as $P(R|R')$ with R' denoting a human-specified reward function that can be sub-optimal. Alternatively, we will also utilize B-REX to create a posterior distribution $P(R|D)$ given demonstrations D.

To approximate R, we assume that the reward function can be generalized as a weighted combination of k features. We define the approximated reward function $r = \Phi w$ where $\Phi \in \mathbb{R}^{SA \times k}$ is the linear feature matrix with rows as state-action pairs and columns as features, and $w \in \mathbb{R}^k$ is the weight vector across k features. We note that rewards at different states can be correlated through k features. These features are defined for each state-action pair in Φ. We also note that features can be nonlinear functions of the given states and actions but can be learned by an auxiliary task. We denote the expected discounted feature counts of policy π as $\mu_\pi \in \mathbb{R}^k$ where $\mu_\pi = \Phi^T u_\pi$. Therefore, the expected return of policy π can be calculated as $\rho(\pi, r) = u_\pi^T \Phi w = \mu_\pi^T w$.

3.3 Measuring Conditional Value at Risk (CVaR)

We use Conditional Value at Risk (CVaR), commonly known as average value of risk and expected tail risk, as our measure of risk. CVaR is convex and a lower bound of Value at Risk (VaR). We use the risk aversion parameter $\alpha \in [0, 1)$ to define VaR_α as the (1-α)-quantile worst case outcome, where $VaR_\alpha[X] = sup\{\ x : P(X \geq x) \geq \alpha\ \}$. We will use $\alpha \in [0.9, 1)$ to test risk sensitive environments. CVaR is defined as:

$$CVaR_\alpha[X] = E[X|X \leq VaR_\alpha[X]] \tag{1}$$

where we assume lower values of CVaR are worse. We prefer using CVaR over VaR as our measurement of risk because VaR is not convex, intractable, and ignores tail risk.

Since we assume lower values of CvaR are worse, we seek to maximize CVaR given the performance of policy π and reward function R. We denote Π as the set of all randomized policies and $\psi : \Pi \times \mathcal{R} \to \mathbb{R}$ as the performance metric for policy π under unknown reward $R \sim P(R)$. Since we seek to find a policy that is robust over the distribution $P(R)$, we optimize CVaR with respect to $\psi(\pi, R) = \rho(\pi, R) = u_\pi^T \Phi w_R$, where w_R characterizes the reward function. Therefore, we aim to find the policy that best fits:

$$\max_{\pi \in \Pi}\ CVaR_\alpha[\psi(\pi, R)] \tag{2}$$

3.4 Linear Programming Formulation

In order to find the policy that solves Equation (2), we note that $\pi : S \to \Delta^A$, the distribution of actions, and that solving for $max_\pi \rho(\pi, R)$, the best policy for a particular reward function, is the same as solving for the linear program:

$$\max_{u \in \mathbb{R}^{SA}}\ (r^T u \mid \sum_{a \in \mathcal{A}} (I - \gamma P_a^T) u^a = p_0, u \geq 0) \tag{3}$$

We denote p_R as the posterior distribution over samples $P(R|D)$. Here, we assume the posterior over reward functions, R, is generated from a set of demonstrations, D. We cannot optimize CVaR

in Equation (1) directly since it is defined over continuous distributions while p_R is discrete. By utilizing the convexity of CVaR to adapt to any distribution, we can rewrite CVaR as:

$$\begin{aligned} CVaR_\alpha[\psi(\pi, R)] &= \max_{\sigma \in \mathbb{R}} \ \sigma - \frac{1}{1-\alpha} p_R^T [\sigma \cdot 1 - \Psi(\pi, R)]_+ \quad where \\ \Psi &= \Big[\psi(\pi, R_1) \quad \psi(\pi, R_2) \quad \dots \quad \psi(\pi, R_m)\Big]^T \end{aligned} \tag{4}$$

Where σ approximates VaR_α, and $[x]_+$ denotes max(0, x). Given Equation (4) under the linear program denoted in Equation (3), the optimal policy can be found in polynomial time by normalizing the state-action occupancies u that best optimize Equation (4). Therefore, the optimal policy π^* can be recovered from optimal occupancies u^* as:

$$\pi^*(s, a) = \frac{u^*(s, a)}{\Sigma_{a' \in \mathcal{A}} u^*(s, a')} \tag{5}$$

3.5 Balancing Expected Return and CVaR

Previously, we found a policy that best maximizes CVaR. To optimize the risk-sensitivity for the outputted policy, we balance expected performance and CVaR with parameter $\lambda \in [0, 1]$. To tune the risk sensitivity of the policy while still maximizing expected return, we define our new objective as:

$$\max_{\pi \in \Pi} \ \lambda E[\psi(\pi, R)] + (1 - \lambda) CVaR_\alpha[\psi(\pi, R)] \tag{6}$$

We note that when $\lambda = 0$, we optimize a policy that maximizes expected return without considering risk. When $\lambda = 1$, we optimize a policy that aims to maximize CVaR and not expected return, creating an optimal risk-neutral Bayesian policy. The finalized BROIL objective is formulated as:

$$\begin{aligned} \max_{u \in \mathbb{R}^{\mathcal{SA}},\, \sigma \in \mathbb{R}} \quad & \lambda p_R^T \Psi(\pi_u, R) + (1 - \lambda)(\sigma - \frac{1}{1-\alpha} p_R^T [\sigma \cdot 1 - \Psi(\pi, R)]_+) \\ \text{s.t.} \quad & \sum_{a \in \mathcal{A}} (I - \gamma P_a^T) u^a = p_0, u \geq 0 \end{aligned} \tag{7}$$

where π_u is the stochastic policy that corresponds to state-action occupancy vector u.

3.6 Adapting BROIL to Continuous Spaces

The previous formulation is only applicable to discrete state and action spaces. In our implementations, we adapted BROIL for continuous spaces using a modified policy gradient. Given a discrete distribution of reward distributions, we estimated ψ using average accumulated rewards given a state for each reward function.

$$\psi(\pi, R_i) \approx \frac{1}{k} \sum_{j=1}^{k} R_\pi^i(\tau_j) \quad and \quad \Psi \approx \Big[\psi(\pi, R_1) \quad \psi(\pi, R_2) \quad \dots \quad \psi(\pi, R_m)\Big]^T \tag{8}$$

Note that $R_\pi^i(\tau_j)$ denotes accumulated reward at the j^{th} rollout calculated from the i^{th} reward function under the current policy π. Ψ is the resultant vector for m reward functions.

Then, using these accumulated rewards, we estimated the values of CVaR and σ, with respect to the reward distribution. Taking advantage of the fact that CVaR is a convex function, we constrained the candidate values of σ to be one of the average aggregated rewards in the distribution of functions.

Then, we could approximate the expected BROIL objective by weighing the aggregated reward in each state by its posterior probability and coefficients in the BROIL objective according to equation 8.

$$Adv_{BROIL}(s) = \sum_{i=1}^{m} P(R^i) V_i^{\pi}(s)(\lambda + 1\{\ \sigma \geq \psi_i\ \} \frac{1-\lambda}{1-\alpha}) \ where\ V_i^{\pi}(s) = E[R_{\pi}^i(\tau) \mid s] \tag{9}$$

Note that $P(R^i)$ is the posterior probability of i^{th} reward function, $V_i^{\pi}(s)$ is a state value estimate of the i^{th} reward function given the agent is at state s, and ψ_i is the i^{th} reward function's expected trajectory returns. Calling these weights Adv_{BROIL}, we replace advantage values in a policy gradient with these values.

$$\begin{aligned} \hat{g}_k = \quad & \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \nabla_{\theta} log \pi_{\theta}(a_t|s_t) Adv_{BROIL}(s_t) \\ & \theta_{k+1} = \theta_k + \alpha_{LR}\ \hat{g}_k \end{aligned} \tag{10}$$

Note that D_k is the set of sampled trajectories, T is an episode's length, α_{LR} is the policy learning rate, and π_{θ} denotes our current policy. We implemented many variations of the above implementation. The above describes our reward-to-go implementation. However, we also adapted this policy gradient to Vanilla Policy Gradients (VPG) and Proximal Policy Optimization (PPO), where we learned a value network that optimized the below objective.

$$\begin{aligned} J_{\phi} = \quad & \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{i}^{T} (V_{\phi}(s_i) - V_{tar}(s_i))^2 \\ & \phi_{k+1} = \phi_k - \alpha_v \nabla J_{\phi} \end{aligned} \tag{11}$$

Note that V_{tar} is calculated using discounted rewards to go. Subsequently, we replaced rewards-to-go with an advantage estimate using GAE-λ as described below to decrease variance:

$$\begin{aligned} \hat{A}_t^{(k)} = \quad & \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^v = -v(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k v(s_{t+k}) \\ & Adv_{GAE,i}^{\pi} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots) \end{aligned} \tag{12}$$

$$Adv_{BROIL}(s) = \sum_{i=1}^{m} P(R^i) Adv_{GAE,i}(\lambda + 1\{\ \sigma \geq \psi_i\ \} \frac{1-\lambda}{1-\alpha}) \tag{13}$$

4 Bayesian Reward Extrapolation (B-REX)

4.1 Background

The MDP environment is defined equivalently for B-REX as it is for BROIL. However, the inputted MDP lacks a reward function (MDP-R). In Bayesian IRL, an algorithm's goal is to extrapolate the latent reward function from a set of demonstration trajectories. Using Boltzmann rationale and defining Q^* as $Q_R^*(s_0, a_0) = E[\sum_{t=0}^{\infty} R(s_t, a_t)|s_0, a_0]$, we attempt to uncover a reward function such that given trajectories have high of executing the following policy:

$$\pi_R^{\beta}(a|s) = \frac{e^{\beta Q_R^*(s,a)}}{\Sigma_{b \in \mathcal{A}} e^{\beta Q_R^*(s,b)}} \tag{14}$$

where R is the true reward function and $\beta \in [0, \infty)$ represents the confidence in the given value function. Given a set of demonstrations $D = \{ \tau_1, \ldots, \tau_m \}$, we define the conditional probability of such demonstrations under the Boltzmann rational given a proposed reward function to be:

$$P(D|R) = \prod_{(s,a)\in D} \pi_R^{\beta}(a|s) = \prod_{(s,a)\in D} \frac{e^{\beta Q_R^*(s,a)}}{\Sigma_{b\in\mathcal{A}} e^{\beta Q_R^*(s,b)}} \tag{15}$$

Similar to other Bayesian IRL methods, new reward proposals are generated using Monte Carlo Markov Chains (MCMC) whose stationary distribution is defined by the posterior probability of reward functions. As a result, Bayesian IRL is typically limited to low-dimensional problems. B-REX attempts to remedy this issue by improving the efficiency of Bayesian IRL methods dramatically.

4.2 Pretraining and Encoding

To make Bayesian IRL adaptable to higher-dimensional environments, B-REX leverages the expressiveness of neural networks. In a pre-training environment, B-REX creates a low-dimensional encoding of the state space. Using inputted trajectories from demonstrations, the algorithm utilizes four self-supervised tasks to create this latent embedding. Firstly, an inverse dynamics model is trained to predict the associated action a_t given the embedding ϕ_t and ϕ_{t+1}. Then, a forward dynamics model is used to predict the future state s_{t+1} from the current local embedding ϕ_t and action a_t. Then, a temporal difference prediction module is trained to learn an embedding ϕ_t that predicts the number of timesteps between two randomly chosen states. Lastly, B-REX train a pixel-to-pixel autoencoder where ϕ_t is the latent encoding.

These various objectives used in making a low-dimensional embedding all encode various aspects of information into the latent state, making it a useful representation for Bayesian IRL after pre-training is finished. In the context of this research endeavor, we skip this step of creating latent encodings and instead directly input state trajectories.

4.3 Bayesian Reward Extrapolation from Ranked Trajectories

Equipped with a latent encoding of our states from demonstrations (or directly inputted state trajectories), B-REX continues to use these trajectories to generate a posterior distribution over reward functions. Given a set of demonstrations D and pairwise preferences over the trajectories $\mathcal{P} = \{ (i, j) : \tau_i < \tau_j \}$, we assume a Boltzmann rational demonstrator and define a pairwise likelihood function[2]:

$$P(D, P|R_\theta) = \prod_{(i,j)\in P} \frac{e^{\beta R_\theta(\tau_j)}}{e^{\beta R_\theta(\tau_i)} + e^{\beta R_\theta(\tau_j)}} \tag{16}$$

where R is the predicted return under a particular reward function and beta measures the confidence in the reward function R. Using the latent encoding as described in the previous section, B-REX represents a reward as a linear function of its encoded features[1], $w^T\phi(s)$. In doing so, B-REX reduces the computation cost of computing a trajectories accumulated reward by caching the summed states of a given trajectory:

$$R_\theta(\tau) = \sum_{s\in\tau} w^T\phi(s) = w^T \sum_{s\in\tau} \phi(s) = w^T\Phi_\tau \tag{17}$$

This results in the following representation of the described likelihood function:

$$P(D, P|R_\theta) = \prod_{(i,j)\in P} \frac{e^{\beta w^T \Phi_{\tau_j}}}{e^{\beta w^T \Phi_{\tau_j}} + e^{\beta w^T \Phi_{\tau_i}}} \tag{18}$$

4.4 Generating New Proposals

After pre-training our latent encoding and caching summed trajectory states, B-REX generates proposals for new reward functions (defined by the vector w) using Monte Carlo Markov Chains (MCMC). We define out proposal for w as follows:

$$w_{t+1} = Normalize(\mathcal{N}(w_t, \sigma)) \tag{19}$$

where σ is a hyperparameter representing proposal width. Using this proposal and equation 16 for acceptance regions, B-REX uses Metropolis Hastings to generate N samples, after appropriate burn-in time.

5 Connecting BROIL and B-REX

This project was intended to be an end-to-end framework for an agent to safely extrapolate a distribution of rewards from demonstration and subsequently return a robust policy that accounts for the distribution's risk-uncertainty. However, due to various limitations in the complexity associated with BROIL and the addition of more environments to test on, this project largely explored BROIL and its ability to adapt to continuous state and action spaces, along with a brief exploration of B-REX afterwards. Instead of generating demonstrations for B-REX to take as input, we made BROIL train many policies of varying optimality and plugged in trajectories from these policies into B-REX. We subsequently analyzed B-REX output distributions to analyze how dissimilar it was to the originally used distribution.

6 Experiments, Results, and Discussion

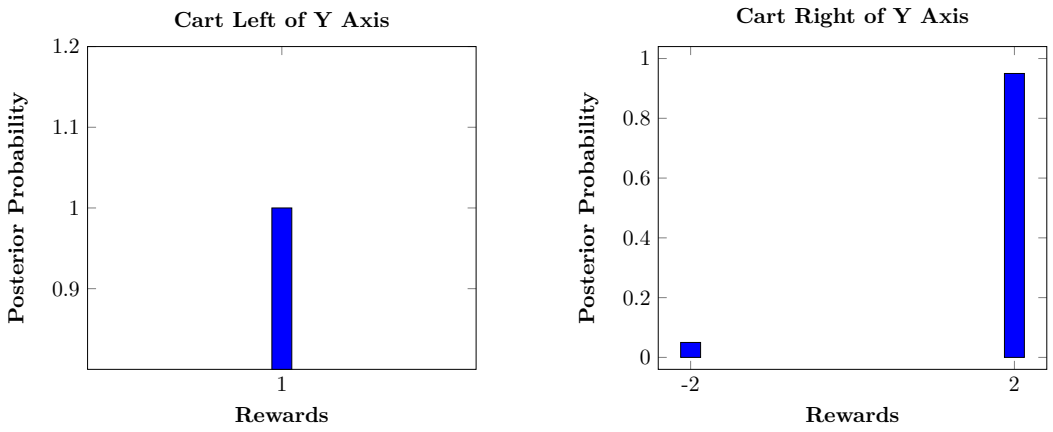
6.1 Environments and Methods

We first conducted tests on the CartPole environment on the BROIL-integrated RTG policy for debugging and benchmarking purposes. Then, we shifted our focus to create reward distributions from PointBot demonstrations using BREX, utilize the reward distribution in BROIL-integrated versions of PPO and VPG policies, and analyze performance regarding risk tolerance and aversion. We also used the same policies to analyze performance in the PointBot environment under a custom reward distribution. From additional feedback, other challenging domains like HalfCheetah and Ant were used to evaluate BROIL-integrated PPO and VPG.

For all BROIL tests, policies with varying levels of $\alpha \in [0, 1)$ and $\lambda \in [0, 1]$ were tested via a grid search and measured for performance with respect to CVaR and true return (as determined by the environment; not our reward distribution). We will hand-craft BROIL reward functions except for the case where BREX will outputs a reward function for the PointBot environment.

6.2 CartPole (BROIL)

We tested the CartPole environment where the objective is for the cart to balance a pendulum. We hand-crafted an uncertain bayesian reward function as input to BROIL as follows:



Note that the expected return is 1.8 when the cart is to the right of the Y-axis and 1 to the left of the Y-axis.

The cart will receive larger reward in expected value when it stays to the right of the Y-axis. However, the cart has a 5% chance of receiving a reward of -2, making a risk-averse policy favor states where the cart is to the left of the Y-axis.

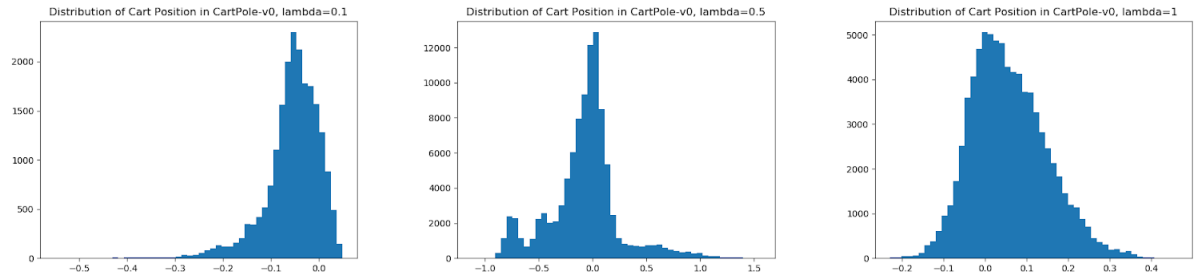


Figure 1: Distribution of Cart Position with $\alpha = 0.95$ under $\lambda = 0.1,\ 0.5$, and 1. The Cart stays to the left of the y-axis more as λ decreases.

By testing the λ values in Figure (1), we note that the cart moves to the left more as λ decreases. This is because by the BROIL objective, a lower λ will prioritize a more risk-averse policy and consider CVaR values more strongly.

Given higher risk-aversion parameter α, CVaR will consider a smaller (1-α)-quantile worst case outcome and become increasingly more risk averse. Figure (2) shows that higher α will lower the expected return of the cart. This is expected since a more risk-averse policy will stay on the left of the Y-axis, choosing a lower return of 1. The cart will purposely choose a lower return for a more robust policy. We note that both higher α and lower λ will increase the safety of the policy while both lower α and higher λ will attempt to maximize the expected value of the policy.

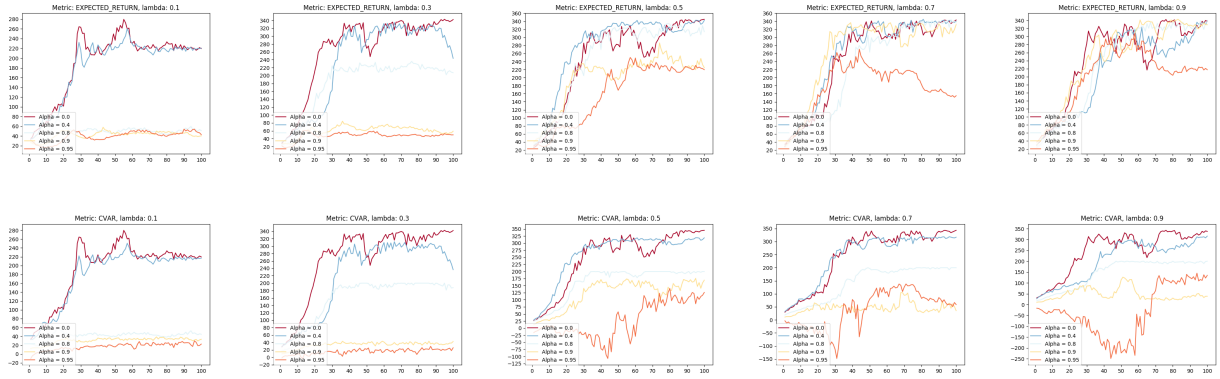


Figure 2: Expected Return and CVaR Values across various distributions of α and λ Values. CVaR value are a lower bound for Expected Return as shown.

6.3 PointBot

The PointBot environment is a custom environment developed by Berkeley's AUTOLab. In this 2D environment, the goal of the point mass is to traverse a small obstacle course and reach the goal location. The point mass is able to go through the obstacle but will receive a reward penalty for doing so (Note: this is with respect to the reward function defined in the environment, not the custom reward distribution we define in which BROIL policies train on). The environment reward function is defined as:

$$Reward = -Distance\ To\ Goal\ State + (Collision\ Cost) * (In\ Obstacle\ Region) \tag{20}$$

"In Obstacle Region" takes on a value of 0 if the point mass is not in the obstacle or 1 if it is. For training under BROIL, our policy will receive a posterior distribution for the "Collision Cost's" value. This distribution will be created either by BREX or hand-crafted. All start states for the point mass will be at (-100, 50) and Goal States will be at (0, 0) with a square obstacle between the states. All PointBot visualizations plot 5 different trajectories of its policy once the policy is done training.

6.3.1 BREX

The BREX framework takes as input various state trajectories in the PointBot environment. These trajectories were taken from varying policies, each of which decided differently whether to pass through the obstacle or travel around it. For each demonstration, we reduced the state space to a two-element one-hot vector denoting whether the agent was inside the obstacle or outside. As a result, calculating σ in equation (17) is calculated easily for each trajectory. Therefore, each MCMC proposal was a two element vector. We hand-labeled and ordered trajectory preferences to feed into the MCMC. To produce our reward distributions, the MCMC was implemented with 200,000 steps, a burn-in of 5,000 steps, and a sampling rate of 20.

Since the outputted proposals were sequences of weight vectors, we discretized the space of reward function vectors by dividing it into 50-by-50 boxes in weight space. As a result, each reward function in the posterior distribution has the following form: (Inside Obstacle Collision Cost, Outside Obstacle Collision Cost) with an associated probability (calculated via Monte Carlo). Note there is also now a cost for being outside an obstacle.

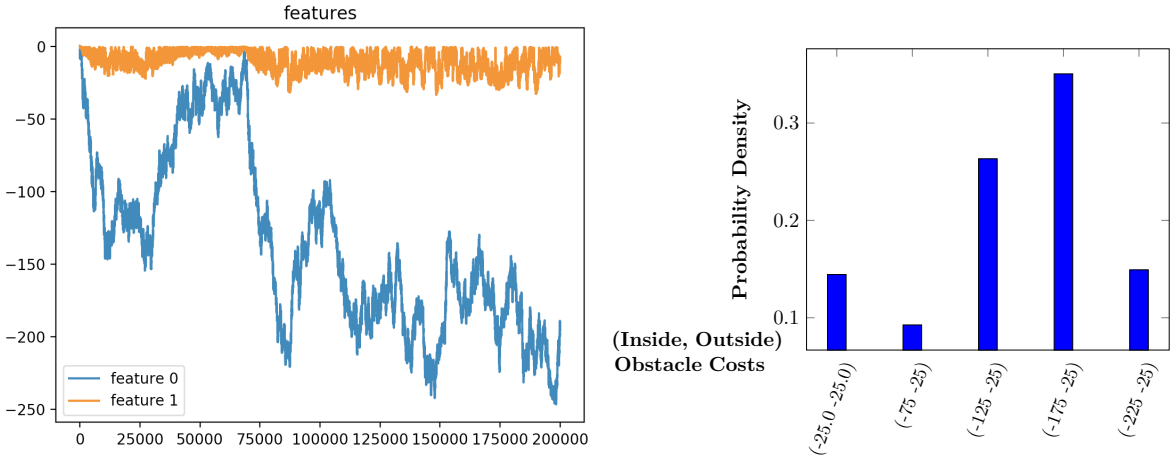


Figure 3: 8 demonstration inputs. 2 demonstrations go through the obstacle and 6 avoid the obstacle. The Monte Carlo Markov Chain is shown Left. The Reward Distribution is shown Right.

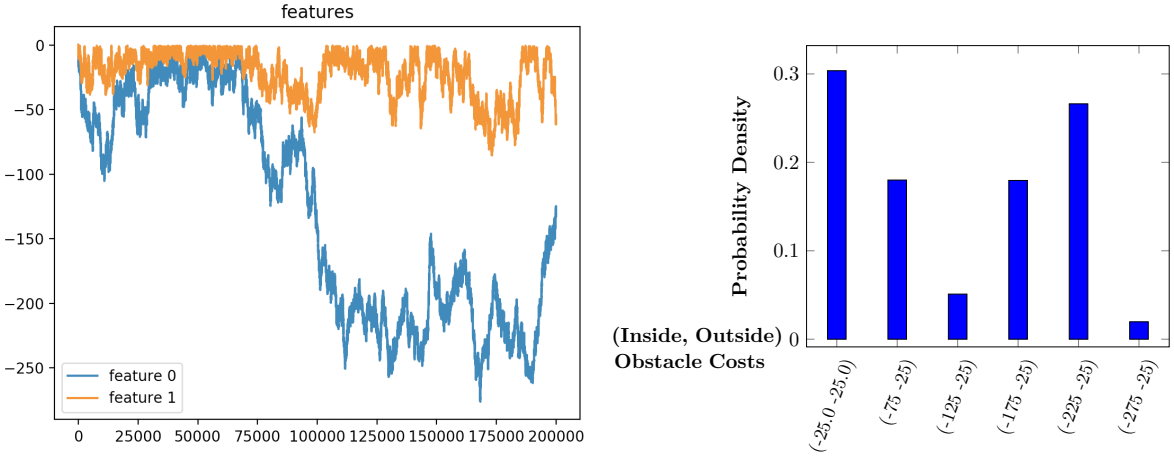


Figure 4: 8 demonstration inputs. 6 demonstrations go through the obstacle and 2 avoid the obstacle. The Monte Carlo Markov Chain is shown Left. The Reward Distribution is shown Right.

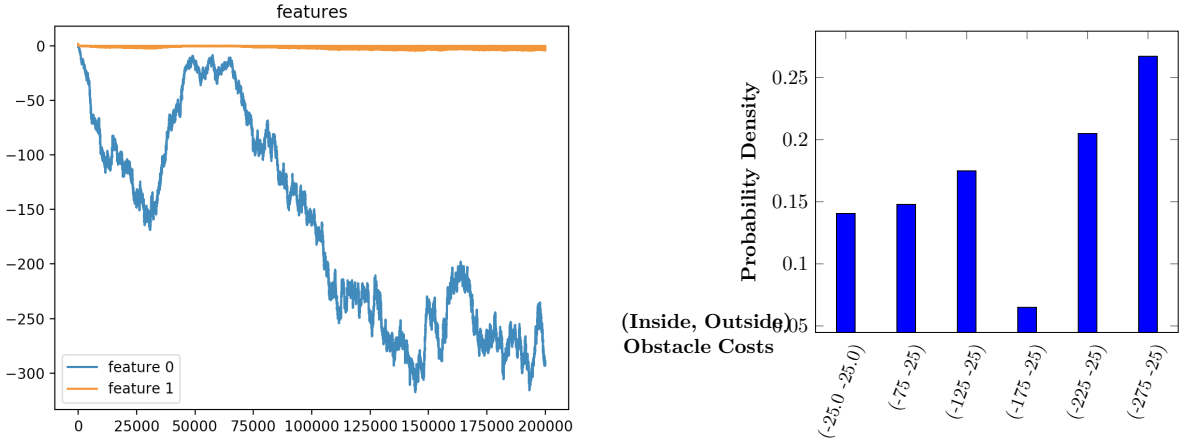


Figure 5: 8 demonstration inputs. 4 demonstrations avoid the obstacle and 4 go through the obstacle. The Monte Carlo Markov Chain is shown Left. The Reward Distribution is shown Right.

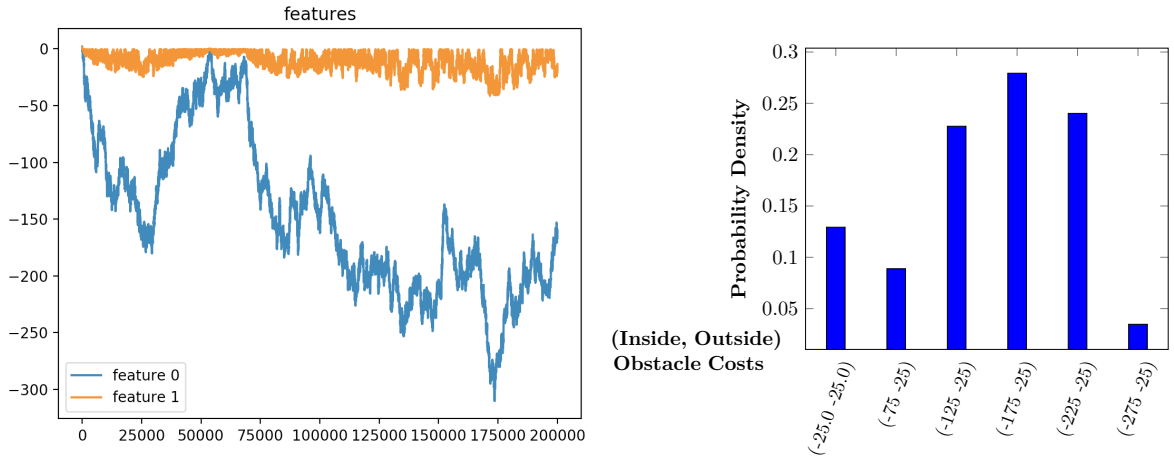


Figure 6: 10 demonstration inputs. 5 demonstrations avoid the obstacle and 5 go through the obstacle. The Monte Carlo Markov Chain is shown Left. The Reward Distribution is shown Right.

We've denoted state feature 0 as "within" and feature 1 as "outside of" obstacles. As a result, it is evident that B-REX extrapolates rewards that penalize travelling inside obstacles. Furthermore, after discretizing weight space, we use Monte Carlo to extract the respective simple distribution over rewards (again representing penalties for being within and outside of obstacles).

It is evident that B-REX is highly sensitive to inputted demonstrations. Not only that, MCMC outputs are relatively high in variance as there is no clear direction in which the distribution is skewed given the number of demonstrations that go through vs. around an obstacle. A large factor contributing to variance is accounted for in the in variance in policies learned in the environment.

6.3.2 BREX to BROIL

Given the reward distribution outputted by BREX in Figure (6), we input this distribution into BROIL and modify the posterior reward function to account for cost outside the obstacle as follows:

$$\begin{aligned} Reward = \quad & -Distance\ To\ Goal\ State + (Inside\ Collision\ Cost) \times (In\ Obstacle\ Region) \\ & + (Outside\ Collision\ Cost) \times (Outside\ Obstacle\ Region) \end{aligned} \tag{21}$$

We then tested this reward distribution with BROIL-integrated PPO and VPG. Because of the very large negative cost associated with "Inside Collision Cost," the PointBot avoids the obstacle given any combination of α and λ parameters for either algorithm. Our results were inconclusive in evaluating BROIL's risk-sensitivity parameters with respect to B-REX's outputted rewards.

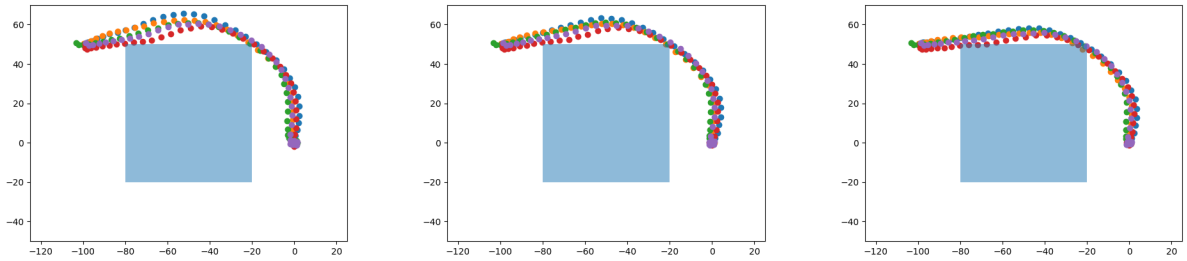


Figure 7: PPO - Visualization across $\alpha = 0.95$ and $\lambda \in (0, 0.5, 1)$ respectively

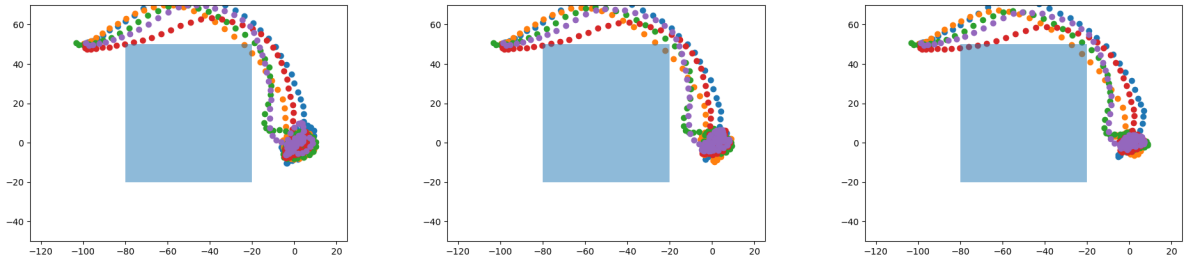
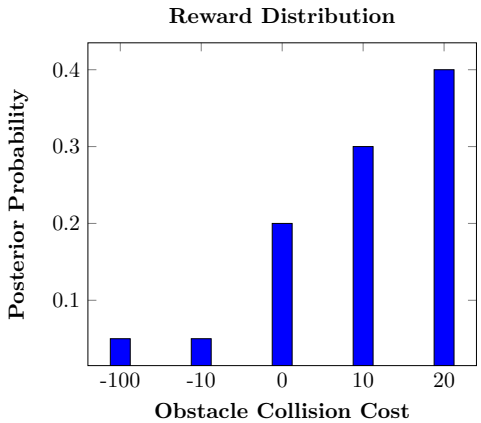


Figure 8: VPG - Visualization across $\alpha = 0.95$ and $\lambda \in (0, 0.5, 1)$ respectively

6.3.3 BROIL

Because the outputted reward function given by BREX heavily penalized states within the obstacle, we were unable to test BROIL's risk sensitivity since a risk-neutral (low α) policy would also avoid the obstacle region. Instead, we handcrafted an obstacle collision cost function to BROIL to analyze more interesting behavior:



The expected return for traveling inside an obstacle is now positive (5.5). However, the PointBot risks a 5% chance of receiving a -10 reward and a 5% chance of receiving a -100 reward, signalling that a risk-averse policy should avoid the obstacle.

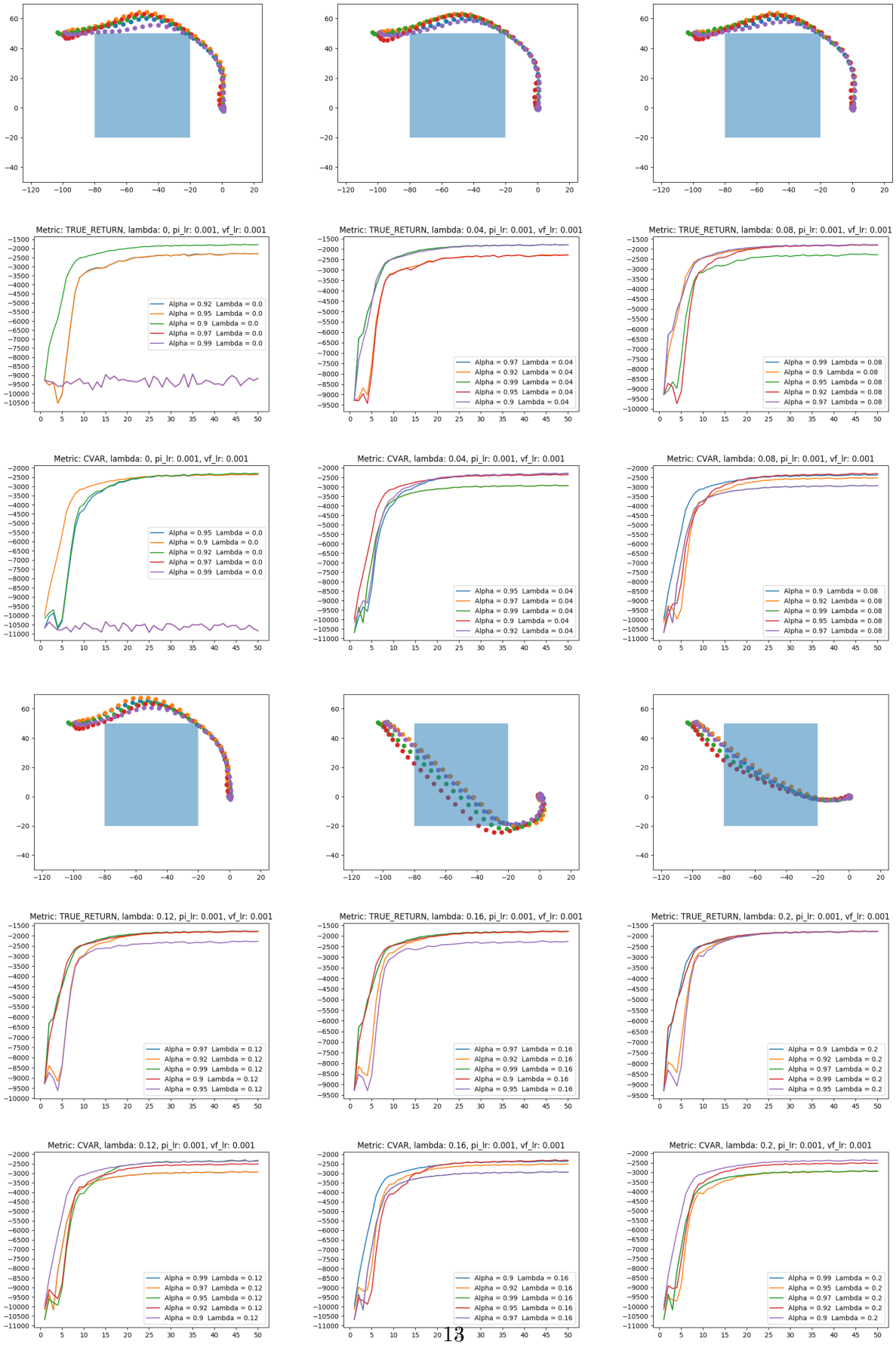


Figure 9: PPO - Visualization ($\alpha = 0.95$), True Return and CVaR values across various distributions of α and $\lambda \in (0, 0.04, 0.08, 0.12, 0.16, 0.2)$ Values.

For PPO, an $\alpha = 0.95$ and $\lambda > 0.12$ begins to enter the obstacle and become less risk-averse. Keeping $\alpha = 0.95$, we discovered that a λ value usually above 0.13 will enter and go straight through the obstacle meaning that $\lambda < 0.13$ will encourage more risk-averse behavior. We note that in the visualization for $\lambda = 0.16$, although the PointBot travels through the obstacle, it exits the obstacle on the lower bottom right side to exit the obstacle in a quicker manner and take a longer distance outside the obstacle to arrive to the goal state. This shows that although the PointBot decides to travel through the obstacle, the policy is still slightly risk-averse. We note that as α gets closer to 0.95, CVaR values decrease since CVaR then corresponds to the 0.05-quantile worst case expected outcome, or a collision cost of -100.

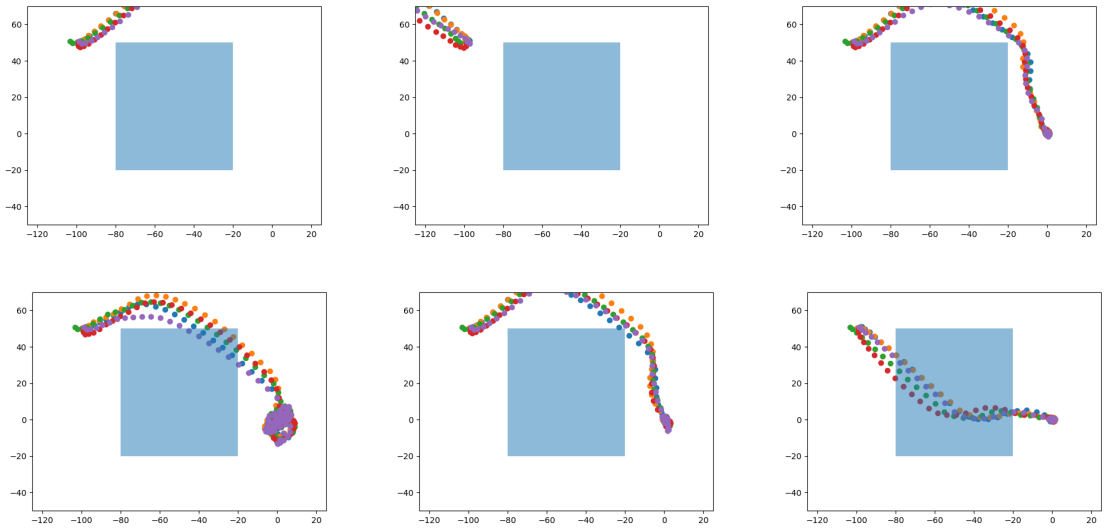


Figure 10: VPG - Visualization across $\alpha = 0.95$ and $\lambda \in (0, 0.04, 0.08, 0.12, 0.16, 0.2)$ respectively

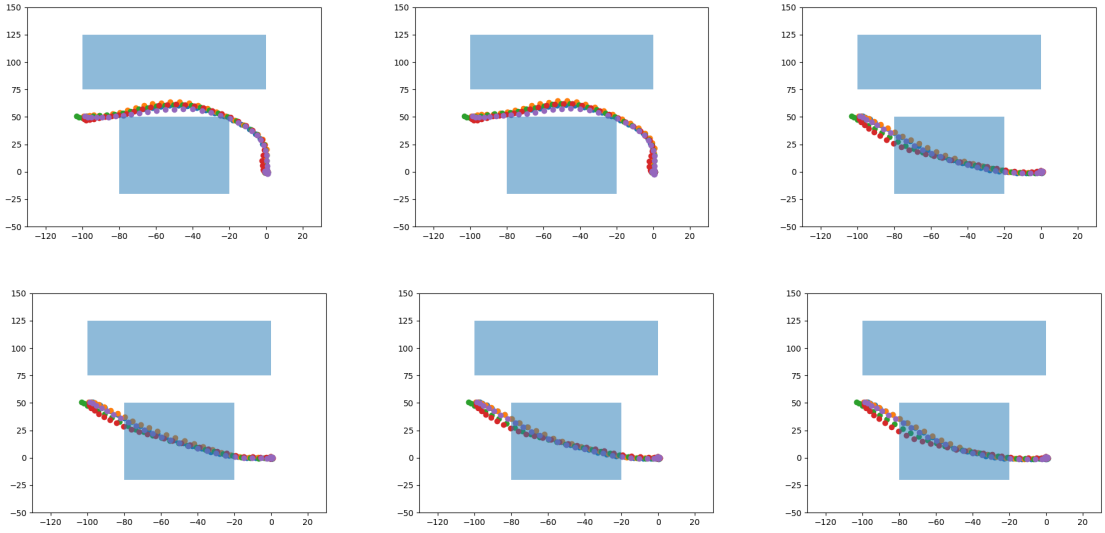
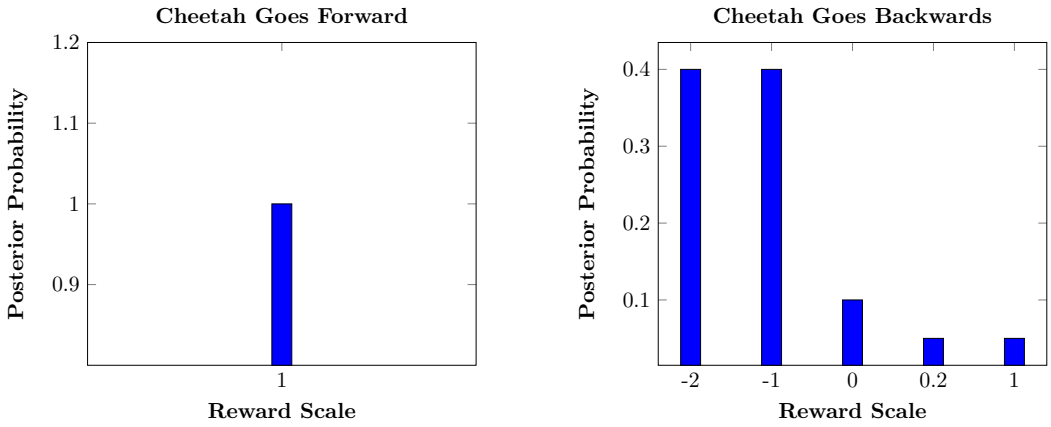


Figure 11: VPG - Additional Obstacle Visualization across $\alpha = 0.95$ and $\lambda \in (0, 0.04, 0.08, 0.12, 0.16, 0.2)$ respectively

After analyzing Figure (10), we noticed many instances when the policy would move in the opposite direction of the goal. This avoidance of the obstacle signals that the policy has become too risk-averse. This is reflected in the policy trajectories that take a long path around the obstacle or even sometimes exit the visualization bounds (plots when $\lambda = 0$ and $\lambda = 0.04$). To constrain such large turns, we placed a second obstacle to constrain the path of our PointBot shown in Figure (11). The PointBot now does not take dramatic turns and instead takes a more direct route to the goal, demonstrating BROIL's risk tolerant properties with the VPG policy.

6.4 Cheetah (BROIL)

The Cheetah environment is located under OpenAI's MuJoCo, and is a continuous control task that makes a 2D cheetah robot run. The custom Bayesian reward function developed for Cheetah scales its velocity reward by varying degrees. Note that more negative scaling when the Cheetah travels backwards yields higher rewards while more positive scaling when Cheetah travels forward yields higher reward. This implies that this reward distribution's expected value encourages running backwards, the opposite intent of the environment. For both Cheetah and Ant environments, this reward represents a possible distribution that IRL could generate when given trajectories that only travel forward. As a result, we test BROIL's performance under reward uncertainty when given states outside of the demonstrations' scopes.



The reward function used for HalfCheetah and Ant was as follows. Note that forward reward represents the velocity of the cheetah or ant.

$$Reward = \begin{cases} forward\ weight * forward\ reward + costs & \text{if forward reward is positive} \\ backward\ weight * forward\ reward + costs & \text{if forward reward is negative} \end{cases} \tag{22}$$

We trained BROIL-integrated PPO for 200 epochs with networks comprised of 2 hidden layers and 64 in width. We trained with batch size of 4000 and updated both value and policy networks for 80 iterations per epoch. A grid search was performed on values of $\alpha = (0, 0.3, 0.5, 0.8, 0.9, 0.95, 0.99)$ and $\lambda \in (0, 0.12, 0.2)$. Unlike the previous environments, the optimal values for α and λ are quite different from CartPole and PointBot. In addition, the highest value of α at 0.99 performed quite poorly.

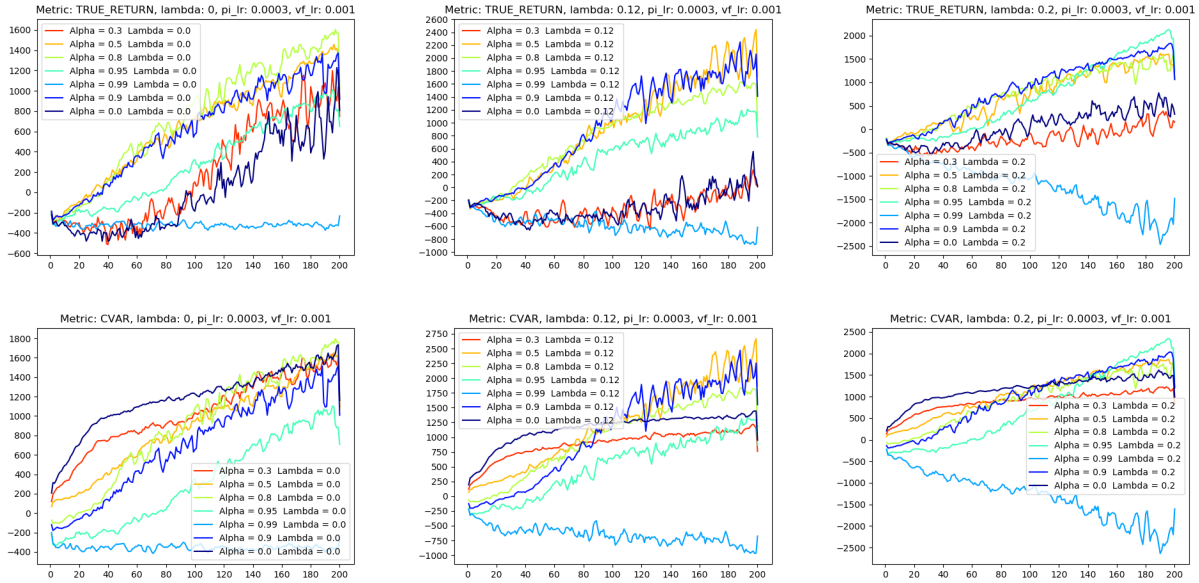
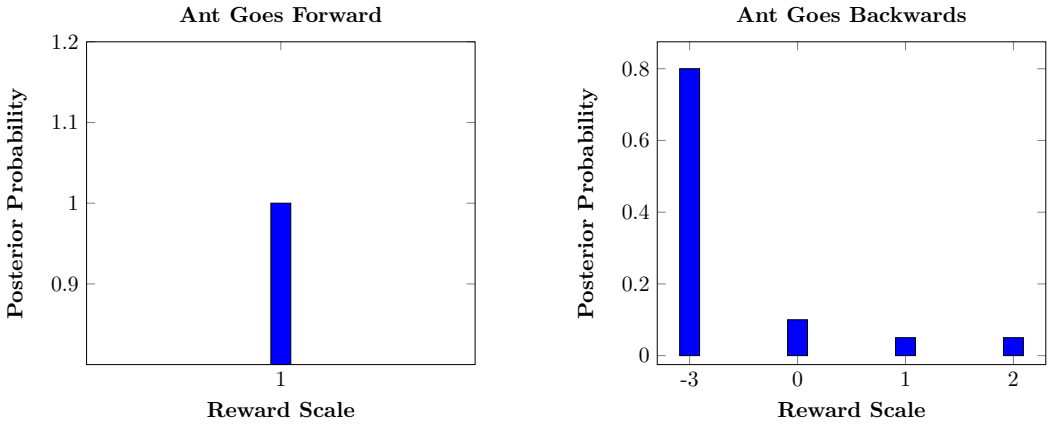


Figure 12: True Return and CVaR across $\alpha = (0, 0.3, 0.5, 0.8, 0.9, 0.95, 0.99)$ and $\lambda \in (0, 0.12, 0.2)$ respectively

Overall, the adaptability of BROIL onto a more challenging environment like HalfCheetah required tuning of a custom reward distribution and function to be quite adaptable as compared to benchmarks. As seen in figure 12, the performance of HalfCheetah can be quite well, but the optimal parameters for α and λ tend to vary per the environment, which makes sense each situation involves a different amount of risk. Values of $\alpha = 0.8$ and 0.9 tend to do quite well, while extremely high or low α values do not. Much of the results here are similar to the Ant environment in terms of optimal hyperparameters, which makes sense given both objectives are quite similar.

6.5 Ant (BROIL)

The Ant environment is also located under OpenAI's MuJoCo, and is similarly a continuous control task that enables a 3D ant robot to run. We implemented a similar Bayesian Reward Function for Ant. Note, the scaling of velocity as described for Cheetah applies here as well.



We trained BROIL-integrated PPO for 200 epochs with networks comprised of 2 hidden layers and 64 in width. We trained with batch size of 4000 and updated both value and policy networks for 80 iterations per epoch.

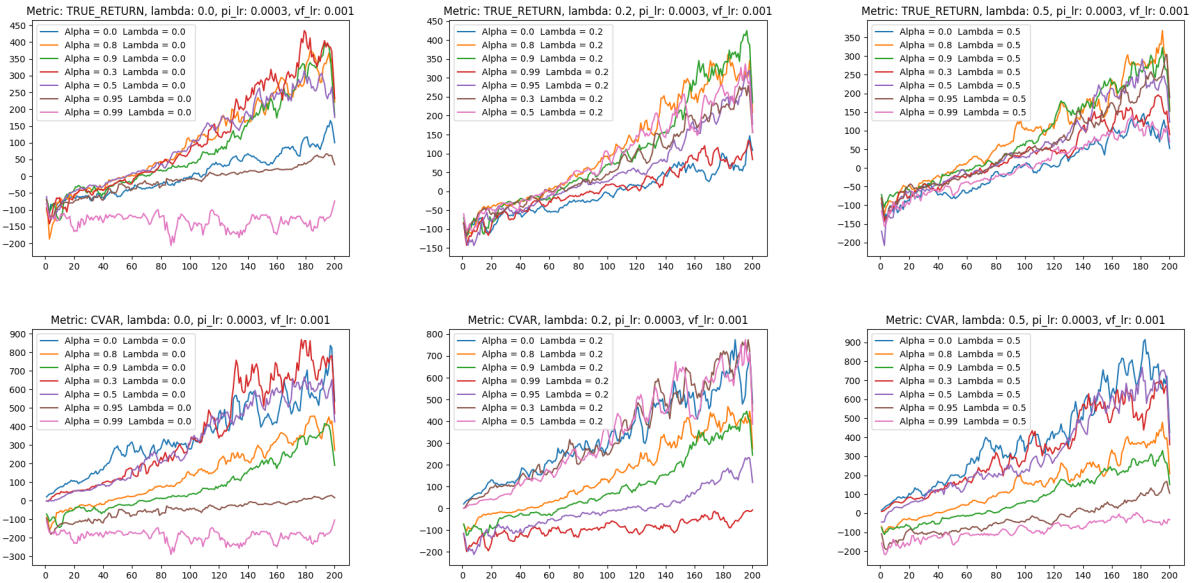


Figure 13: True Return and CVaR across $\alpha = (0, 0.3, 0.5, 0.8, 0.9, 0.95, 0.99)$ and $\lambda \in (0, 0.2, 0.5)$ respectively

In both Cheetah and Ant environments, we can clearly see policy performance changes with respect to α and λ. The policy attains worse performance for low α as optimizations more closely resembles the maximization of expected return of posterior reward, which encourages the cheetah or ant to travel backwards. Furthermore, as we increase α, the policy attains better performance as optimizations reflect risk-averse updates, discouraging either agent to travel backwards. However, increasing α past a specific tolerance (0.95) tends to worsen performance due to numerical instability issues in computing the policy gradient. Similarly, higher values of λ have worse performance as updates more directly optimize expected return over the posterior.

7 Conclusions

In this work, we presented an end-to-end framework for risk-tolerant learning from demonstration. We found that adapting BROIL to continuous action spaces, higher dimensional problems and more complex environments was relatively successful in its response to risk aversion. Furthermore, we witnessed the success of our learning from demonstration framework by analyzing B-REX's reasonable reward distribution extrapolations followed by BROIL's policy optimization under such uncertainty. In future works, we hope to include higher dimensional environments, such as learning from pictures, and using latent embeddings as state inputs into B-REX.

8 Acknowledgements and Contributions

We wish to acknowledge the University of California, Berkeley CS285 Fall 2020 teaching team for providing us with guidance and support for this research. We appreciate Professor Sergey Levine and the instructors Michael Janner, Vitchyr Pong, Aviral Kumar, and Sasha Khazatsky for their continuous feedback on the status of this research and the providing new ideas for additional exploration areas. We would also like to especially acknowledge Daniel Brown for his contribution to the initial BROIL and B-REX algorithms as well as guidance during weekly meetings. Finally, we wish to acknowledge the NeurIPS 2020 Conference for providing a sample template for this paper which could be reproduced with permission. Resources for this research include OpenAI, MuJoCo, and Google Colab, which were all critical in the research process.

For this research, each team member participated in additional weekly meetings to update on progress with each other. Together, analysis of results and pivot directions were taken in team discussions to achieve optimal output of quality results. Coordination of environments and comparison methods were discussed extensively. Additional discussion on B-REX and BROIL theory was done at nearly every meeting to ensure complete understanding. Contributions for each team member include:

Austin Nguyen - Pioneered the exploration of different reward functions for each environment and led the development for results on the Ant environment. Led adaptation of BROIL to continuous space and PPO. Wrote scripts for plotting results for easier analysis. Implemented reward extrapolation connection from B-REX to BROIL.

Jerry Zhu - Led the development for results from the PointBot environment as well as coordinated team meetings. Served as the primary point of contact with Daniel Brown. Implemented changes to PointBot environment for additional testing. Gathered trajectories and plugged into B-REX for reward extrapolation.

Peter Zhu - Led the development for results from the Cheetah environment as well as moved local experiment runs onto Google Colab integrated with MuJoCo. Developed new environment integration methods for code base. Experimented with varying reward distributions. Became LaTeX expert.

9 References

[1] Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the 21st International Conference on Machine Learning, 2004.

[2] Bradley, R. A. and Terry, M. E. Rank analysis of incomplete block designs: I. the method of paired comparisons.Biometrika, 39(3/4):324–345, 1952.

[3] Brown D., Coleman R., Srinivasan R., and Niekum S., Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences. arXiv:2002.09089, 2020

[4] Brown, D., Niekum S., and Petrik M., Bayesian Robust Optimization for Imitation Learning. arXiv:2007.12315, 2020

[5] Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In International Conference on Machine Learning, 2016b.

[6] Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. Inverse reward design. In Advances in neural information processing systems, pp. 6765–6774, 2017.

[7] Khan, M. E., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. Fast and scalable bayesian

deep learning by weight-perturbation in adam. arXiv:1806.04854, 2018.

[8] Laskey, M., Lee, J., Fox, R., Dragan, A., and Goldberg, K. Dart: Noise injection for robust imitation learning. Conference on Robot Learning (CoRL), 2017.

[9] Petrik, M. and Russell, R. H. Beyond confidence regions: Tight bayesian ambiguity sets for robust mdps. arXiv:1902.07605, 2019.

[10] Ramachandran, D. and Amir, E. Bayesian inverse reinforcement learning. In Proceedings of the 20th International Joint Conference on Artifical intelligence, pp. 2586–2591, 2007.

[11] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. arXiv:1707.06347, 2017.

[12] Thananjeyan, B., Balakrishna, A., Rosolia, U., Li, F., McAllister, R., Gonzalez, J. E., Levine, S., Borrelli, F., and Goldberg, K. Safety augmented value estimation from demonstrations (SAVED): Safe deep model based RL for sparse cost robotic tasks. arXiv:1905.13402, 2019.