Robust Reinforcement Learning using Adversarial Populations

Eugene Vinitsky[*]
Department of Mechanical Engineering
UC Berkeley
evinitsky@berkeley.edu

Yuqing Du[*]
Department of EECS
UC Berkeley
yuqing_du@berkeley.edu

Kanaad Parvate[*]
Department of EECS
UC Berkeley
kanaad@berkeley.edu

Kathy Jang
Department of EECS
UC Berkeley
kathyjang@berkeley.edu

Pieter Abbeel
Department of EECS
UC Berkeley
pabbeel@cs.berkeley.edu

Alexandre Bayen
Department of EECS
Institute of Transportation Studies
UC Berkeley
bayen@berkeley.edu

Abstract

Reinforcement Learning (RL) is an effective tool for controller design but can struggle with issues of robustness, failing catastrophically when the underlying system dynamics are perturbed. The Robust RL formulation tackles this by adding worst-case adversarial noise to the dynamics and constructing the noise distribution as the solution to a zero-sum minimax game. However, existing work on learning solutions to the Robust RL formulation has primarily focused on training a single RL agent against a single adversary. In this work, we demonstrate that using a single adversary does not consistently yield robustness to dynamics variations under standard parametrizations of the adversary; the resulting policy is highly exploitable by new adversaries. We propose a population-based augmentation to the Robust RL formulation in which we randomly initialize a population of adversaries and sample from the population uniformly during training. We empirically validate across robotics benchmarks that the use of an adversarial population results in a more robust policy that also improves out-of-distribution generalization. Finally, we demonstrate that this approach provides comparable robustness and generalization as domain randomization on these benchmarks while avoiding a ubiquitous domain randomization failure mode.

1 Introduction

Developing controllers that work effectively across a wide range of potential deployment environments is one of the core challenges in engineering. The complexity of the physical world means that the models used to design controllers are often inaccurate. Optimization based control design approaches, such as reinforcement learning (RL), have no notion of model inaccuracy and can lead to controllers that fail catastrophically under mismatch. In this work, we aim to demonstrate an effective method for

[*] Equal authorship

Preprint. Under review.

training reinforcement learning policies that are robust to model inaccuracy by designing controllers that are effective in the presence of worst-case adversarial noise in the dynamics.

One effective approach to induce robustness has been domain randomization [27, 8], a method where a designer with expertise identifies the components of the model that they are uncertain about. They then construct a set of training environments where the uncertain components are randomized, ensuring that the agent is robust on average to this set. However, this requires careful parametrization of the uncertainty set as well as hand-designing of the environments.

A more easily automated approach is to formulate the problem as a zero-sum game and learn an adversary that perturbs the transition dynamics [26, 9, 18]. If a global Nash equilibrium of this problem is found, then that equilibrium provides a worst case performance bound under the specified set of perturbations. Besides the benefit of removing user design once the perturbation mechanism is specified, this approach is maximally conservative, which is useful for safety critical applications.

However, the aforementioned literature on learning an adversary predominantly uses a single stochastic adversary. This raises a puzzling question: the minimax problem does not necessarily have any pure Nash equilibria (see Appendix C [26]) but the existing robust RL literature mostly appears to attempt to solve for pure Nash equilibria. That is, the most general form of the minimax problem searches over distributions of adversary and agent policies

$$\max_{p \in \mathcal{P}(\Theta)} \min_{q \in \mathcal{Q}(\Omega)} \mathbb{E}_{\theta \sim p} \left[\mathbb{E}_{\omega \sim q} \left[h(\theta, \omega) \right] \right] \tag{1}$$

where $\mathcal{P}(\Theta), \mathcal{Q}(\Omega)$ are distributions over policies and $h(\theta, \omega)$ is a score function (for example, expected cumulative reward). However, this problem is approximated in the literature by the fixed-policy problem

$$\max_{\theta} \min_{\omega} h(\theta, \omega) \tag{2}$$

We contend that this reduction to a single adversary approach can sometimes fail to result in improved robustness under standard parametrizations of the adversary policy.

The following example provides some intuition for why using a single adversary can decrease robustness. Consider a robot trying to learn to walk east-wards while an adversary outputs a force representing wind coming from the north or the south. For a fixed, deterministic adversary the agent knows that the wind will come from either south or north and can simply apply a counteracting force at each state. Once the adversary is removed, the robot will still apply the compensatory forces and possibly become unstable. Stochastic Gaussian policies (which are ubiquitous in continuous control) offer little improvement: low entropy policies can be counteracted whereas high entropy policies would endow the robot with the prior that the wind cancels on average. Under these standard policy parametrizations, which cannot represent a distribution over policies, we cannot use an adversary to endow the agent with a prior that a persistent, strong wind could come either from north or south. This leaves the agent exploitable to this class of perturbations.

The use of a single adversary in the robustness literature is in contrast to the multi-player game literature. In multi-player games, large sets of adversaries are used to ensure that an agent cannot easily be exploited [29, 5, 3]. Drawing inspiration from this literature, we introduce **RAP** (Robustness via Adversary Populations): a randomly initialized population of adversaries that we sample from at each rollout and train alongside the agent. Returning to our example of a robot perturbed by wind, if the robot learns to cancel any one of the adversaries effectively, then that opens a niche for an adversary to exploit by applying forces in another direction. As the number of adversaries increases, the robot is eventually endowed with the prior that a strong wind could come from either direction and that it must walk carefully to avoid being toppled over.

Our contributions are as follows:

- Using a set of continuous control tasks, we provide evidence that a single adversary does not have a consistent positive impact on the robustness of an RL policy while the use of an adversary population provides improved robustness across all considered examples.
- We investigate the source of the robustness and show that the single adversary policy is exploitable by new adversaries whereas policies trained with RAP are robust to new adversaries.
- We demonstrate that adversary populations can be competitive with domain randomization while avoiding potential failure modes of domain randomization.

2 Related Work

This work builds upon robust control [30], a branch of control theory focused on finding optimal controllers under worst-case perturbations of the system dynamics. The Robust Markov Decision Process (R-MDP) formulation extends this worst-case model uncertainty to uncertainty sets on the transition dynamics of an MDP and demonstrates that computationally tractable solutions exist for small, tabular MDPs [16, 12]. For larger or continuous MDPs, one successful approach has been to use function approximation to compute approximate solutions to the R-MDP problem [25].

One prominent variant of the R-MDP literature is to interpret the perturbations as an adversary and attempt to learn the distribution of the perturbation under a minimax objective. Two variants of this idea that tie in closely to our work are Robust Adversarial Reinforcement Learning (RARL) [18] and and Noisy Robust Markov Decision Processes (NR-MDP) [26] which differ in how they parametrize the adversaries: RARL picks out specific robot joints that the adversary acts on while NR-MDP adds the adversary action to the agent action. Both of these works attempt to find an equilibrium of the minimax objective using a single adversary; in contrast our work uses a large set of adversaries and shows improved robustness relative to a single adversary.

An alternative to the minimax objective, domain randomization, asks a designer to explicitly define a distribution over environments that the agent should be robust to. For example, [17] varies simulator friction, mass, table height, and controller gain (along with several other parameters) to train a robot to robustly push a puck to a target location in the real world; [1] added noise to friction and actions to transfer an object pivoting policy directly from simulation to a Baxter robot. Additionally, domain randomization has been successfully used to build accurate object detectors solely from simulated data [27], to zero-shot transfer a quadcopter flight policy from simulation [21].

However, as we discuss in Sec. 6, a policy that performs well on average across simulation domains is not necessarily robust as it may trade off performance on one set of parameters to maximize performance in another. EPOpt [20] addresses this by replacing the uniform average across distributions with the conditional value at risk (CVaR) [4, 24] a soft version of the minimax objective in which the optimization is only performed over a small percentage of the worst performing parameters. This is an interesting approach to align the domain randomization objective with the minimax objective and could be made compatible with our approach by only training using a subset of the strongest adversaries.

Our demonstration of overfitting to a single adversary is not new; there is extensive work establishing that agents trained independently in multi-agent settings can result in non-robust policies. [6] show that in zero-sum games adversary pairs trained via RL are not robust to replacement of the adversary with a different adversary policy. [10] extends this idea to general sum games by training a population of agent-agent pairs and showing that taking two pairs and swapping the agents in them leads to failure to accomplish the objective. [23] establishes that even in tabular settings (in this case, a general-sum version of Rock Paper Scissors), iterated best response to pure Nash strategies can lead to cyclical behavior and a failure to converge to equilibrium.

The use of population based training is also a standard technique in multi-agent settings. Alphastar, the grandmaster-level Starcraft bot, uses a population of "exploiter" agents that fine-tune against the bot to prevent it from developing exploitable strategies [29]. [5] establishes a set of sufficient geometric conditions on games under which the use of multiple adversaries will ensure gradual improvement in the strength of the agent policy. They empirically demonstrate that learning in games can often fail to converge without populations. Finally, Active Domain Randomization [14] is a very close approach to ours, as they use a population of adversaries to select domain randomization parameters whereas we use a population of adversaries to directly perturb the agent actions. Additionally, they use a Stein Variation Policy Gradient [13] to ensure diversity in their adversaries and a discriminator reward instead of a minimax reward whereas our work does not have any explicit coupling between the adversary gradient updates and uses a simpler zero-sum reward function.

3 Background

3.1 Notation.

In this work we use the framework of a multi-agent, finite-horizon, discounted, Markov Decision Process (MDP) [19] defined by a tuple $\langle A_{\text{agent}} \times A_{\text{adversary}}, S, \mathcal{T}, \mathcal{R}, \gamma \rangle$. Here A_{agent} is the set of actions for the agent, $A_{\text{adversary}}$ is the set of actions for the adversary, S is a set of states, $\mathcal{T} : A_{\text{agent}} \times A_{\text{adversary}} \times S \rightarrow \Delta(S)$ is a transition function, $R : A_{\text{agent}} \times A_{\text{adversary}} \times S \rightarrow \mathbb{R}$ is a reward function and γ is a discount factor. S is shared between the adversaries as they share a state-space with the agent. The goal for a given MDP is to find a policy π_θ parametrized by θ that maximizes the expected cumulative discounted reward $J^\theta = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | \pi_\theta\right]$. The conditional in this expression is a short-hand to indicate that the actions in the MDP are sampled via $a_t \sim \pi_\theta(s_t, a_{t-1})$. We denote the agent policy parametrized by weights θ as π_θ and the policy of adversary i as $\bar{\pi}_{\phi_i}$. Actions sampled from the adversary policy $\bar{\pi}_{\phi_i}$ will be written as $\bar{a}_t^i$. We use ξ to denote the parametrization of the system dynamics (e.g. different values of friction, mass, wind, etc.) and the system dynamics for a given state and action as $s_{t+1} \sim f_\xi(s_t, a_t)$.

3.2 Baselines

Here we outline prior work and the approaches that will be compared with RAP. Our baselines consist of a single adversary and domain randomization.

3.2.1 Single Minimax Adversary

Our adversary formulation uses the *Noisy Action Robust MDP* [26] in which the adversary adds its actions onto the agent actions. The objective is

$$\max_\theta \min_\phi \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t + \alpha \bar{a}_t) | \pi_\theta,\ \bar{\pi}_\phi\right] \tag{3}$$

where α is a hyperparameter controlling the adversary strength.

We note two important restrictions inherent to this adversarial model. First, since the adversary is only able to attack the agent through the actions, there is a restricted class of dynamical systems that it can represent; this set of dynamical systems may not necessarily align with the set of dynamical systems that the agent may be tested in. This is simply a restriction caused by the choice of adversarial perturbation and could be alleviated by using different adversarial parametrizations e.g. perturbing the transition function directly.

In addition to the restricted set of dynamical systems that the NR-MDP can represent, there is a limitation induced by standard RL agent parametrizations. In particular, agents are often parametrized by either having deterministic actions or having their actions drawn from a probability distribution (i.e. we pass a state through our policy, it outputs parameters of a distribution and we sample the actions from that distribution). The single adversary cannot represent all the systems that we intend the agent to be robust to as a consequence of the parametrization. For example, suppose the agent is currently at some state s_t and the adversary outputs an action $\bar{a}_t$. In the deterministic case, the agent knows that the adversary will never output $-\bar{a}_t$ even though $-\bar{a}_t$ is clearly in the class of possible perturbations.

3.2.2 Dynamics Randomization

Domain randomization is the setting in which the user specifies a set of environments which the agent should be robust to. This allows the user to directly encode knowledge about the likely deviations between training and testing domains. For example, the user may believe that friction is hard to measure precisely and wants to ensure that their agent is robust to variations in friction; they then specify that the agent will be trained with a wide range of possible friction values. We use ξ to denote some vector that parametrizes the set of training environments (e.g. friction, masses, system dynamics, etc.). We denote the domain over which ξ is drawn from as Ξ and use $\mathcal{P}(\Xi)$ to denote some probability distribution over ξ. The domain randomization objective is

$$\begin{gathered} \max_\theta \mathbb{E}_{\xi \sim \mathcal{P}(\Xi)}\left[\mathbb{E}_{s_{t+1} \sim f_\xi(s_t, a_t)}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | \pi_\theta\right]\right] \\ s_{t+1} \sim f_\xi(s_t, a_t) \\ a_t \sim \pi_\theta(s_t) \end{gathered} \tag{4}$$

Here the goal is to find an agent that performs well on average across the distribution of training environment. Most commonly, and in this work, the parameters ξ are sampled uniformly over Ξ.

4 RAP: Robustness via Adversary Populations

RAP extends the minimax objective with a population based approach. Instead of a single adversary, at each rollout we will sample uniformly from a population of adversaries. By using a population, the agent is forced to be robust to a wide variety of potential perturbations instead of a single perturbation. If the agent begins to overfit to any one adversary, this opens up a potential niche for another adversary to exploit. For problems with only one failure mode, we expect the adversaries to all come out identical to the minimax adversary, but as the number of failure modes increases the adversaries should begin to diversify to exploit the agent. To induce this diversity, we will rely on randomness in the gradient estimates and randomness in the initializations of the adversary networks rather than any explicit term that induces diversity. While the idea of using populations does not preclude explicit terms in the loss to encourage diversity, we find that our chosen sources of diversity are sufficient for our purposes.

Denoting $\bar{\pi}_{\phi_i}$ as the i-th adversary and $i \sim U(1, n)$ as the discrete uniform distribution defined on 1 through n, the objective becomes

$$\begin{gathered}\max_{\theta} \min_{\phi_1} \ldots \min_{\phi_n} \;\; \mathbb{E}_{i \sim U(1,n)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, \alpha \bar{a}_t^i) | \pi_\theta,\ \bar{\pi_{\phi_i}} \right] \\ s_{t+1} \sim f(s_t, a_t + \alpha \bar{a}_t)\end{gathered} \tag{5}$$

For a single adversary, this is equivalent to the *minimax adversary* described in Sec. 3.2.1

We will optimize this objective by converting the problem into the equivalent zero-sum game. At the start of each rollout, we will sample an adversary index from the uniform distribution and collect a trajectory in using the agent and the selected adversary. For notational simplicity, we assume the trajectory is of length M and that adversary i will participate in J_i total trajectories while, since the agent participates in every rollout, it will receive J total trajectories. We denote the j-th collected trajectory for the agent as $\tau_j = (s_0, a_0, r_0, s_1) \times \cdots \times (s_M, a_M, r_M, s_{M+1})$ and the associated trajectory for adversary i as $\tau_j^i = (s_0, a_0, -r_0, s_1) \times \cdots \times (s_M, a_M, -r_M, s_M)$. Note that the adversary reward is simply the negative of the agent reward.

We will use Proximal Policy Optimization [22] (PPO) to update our policies. We caution that we have overloaded notation slightly here and for adversary i, $\tau_{j=1:J_i}^i$ refers only to the trajectories in which the adversary was selected: adversaries will only be updated using trajectories where they were active. At the end of a training iteration, we update all our policies using gradient descent. The algorithm is summarized below:

Algorithm 1: Robustness via Adversary Populations

```
Initialize $\theta, \phi_1 \cdots \phi_n$ using Xavier initialization [7];
while not converged do
    for rollout j=1...J do
        sample adversary $i \sim U(1, n)$;
        run policies $\pi_\theta, \bar{\pi_{\phi_i}}$ in environment until termination;
        collect trajectories $\tau_j, \tau_j^i$
    end
    update $\theta, \phi_1 \cdots \phi_n$ using PPO [22] and the collected trajectories;
end
```

We call the agent trained to optimize this objective using Algorithm 1 the *RAP agent*.

5 Experiments

In this section we present experiments on continuous control tasks from the OpenAI Gym Suite [2, 28]. We compare with our baselines and evaluate the efficacy of a population of learned adversaries across a wide range of state and action space sizes. We investigate the following hypotheses:

H1. Agents are more likely to overfit to a single adversary than a population of adversaries, leaving them more exploitable on in-distribution tasks.

H2. Agents trained against a population of adversaries will generalize better, leading to improved performance on out-of-distribution tasks.

H3. Naive parametrization of domain randomization can result in a brittle policy, even when evaluated on the same distribution it was trained on.

H4. While a larger adversary populations can represent more varied dynamics, there will be diminishing returns due to the decreased environment steps each adversary receives.

In-distribution tasks refer to the agent playing against perturbations that are in the training distribution: adversaries that add their actions onto the agent. However, the particular form of the adversary and their restricted perturbation magnitude means that there are many dynamical systems that they cannot represent (for example, significant variations of joint mass and friction). These tasks are denoted as out of distribution tasks. All of the tasks in the test set described in Sec. 5.1 are likely out-of-distribution tasks.

5.1 Experimental Setup and Hyperparameter Selection

While we provide exact details of the hyperparameters in the Appendix, adversarial settings require additional complexity in hyperparameter selection. In the standard RL procedure, optimal hyperparameters are selected on the basis of maximum expected cumulative reward. However, if an agent playing against an adversary achieves a large cumulative reward, it is possible that the agent was simply playing against a weak adversary. Conversely, a low score does not necessarily indicate a strong adversary nor robustness: it could simply mean that we trained a weak agent.

To address this, we adopt a version of the train-validate-test split from supervised learning. We use the mean policy performance on a suite of validation tasks to select the hyperparameters, then we train the policy across ten seeds and report the resultant mean and standard deviation over twenty trajectories. Finally, we evaluate the seeds on a holdout test set of eight additional model-mismatch tasks. These tasks vary significantly in difficulty; for visual clarity we report only the average across tasks in this paper and report the full breakdown across tasks in the Appendix.

We experiment with the Hopper, Ant, and Half-Cheetah continuous control environments shown in Fig. 1. To generate the validation model mismatch, we predefine ranges of mass and friction coefficients as follows: for Hopper, mass $\in [0.7, 1.3]$ and friction $\in [0.7, 1.3]$; Half Cheetah and Ant, mass $\in [0.5, 1.5]$ and friction $\in [0.1, 0.9]$. We scale the friction of every Mujoco geom and the mass of the torso with the same (respective) coefficients. We compare the robustness of agents trained via RAP against: 1) agents trained against a single adversary in a zero-sum game, 2) agents trained using domain randomization, and 3) an agent trained only using PPO and no perturbation mechanism. To train the domain randomization oracle, at each rollout we uniformly sample a friction and mass coefficient from the validation set ranges. We then scale the friction of all geoms and the mass of the torso by their respective coefficients; this constitutes directly training on the validation set which creates a strong baseline. To generate the test set of model mismatch, we take both the highest and lowest friction coefficients from the validation range and apply them to different combinations of individual geoms. For the exact selected combinations, please refer to Appendix B.

5.2 Computational Details and Reproducibility

All of our experiments are run on c4.8xlarge 36 vCPU instances on AWS EC2. Our full paper can be reproduced for a cost of $\approx$ \$100 (full breakdown in Appendix) but we provide all of the trained policies, data, and code at `https://github.com/eugenevinitsky/robust_RL_multi_adversary` to simplify reproducibility. For our RL algorithms we use the RLlib 0.8.0 [11] implementation of PPO [22]. For exact hyperparameters, please refer to the Appendix. Since both gradient computations and forwards passes can be batched across the adversaries, there is no additional run-time cost relative to using a single adversary.

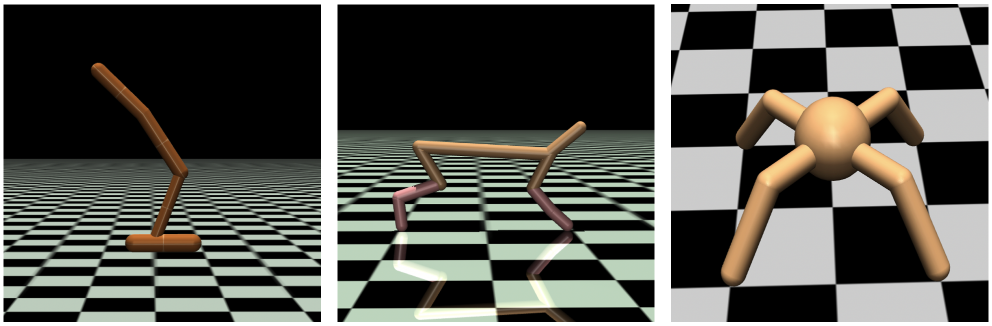


Figure 1: From left to right, the Hopper, Half-Cheetah, and Ant environments we use to validate our approaches.

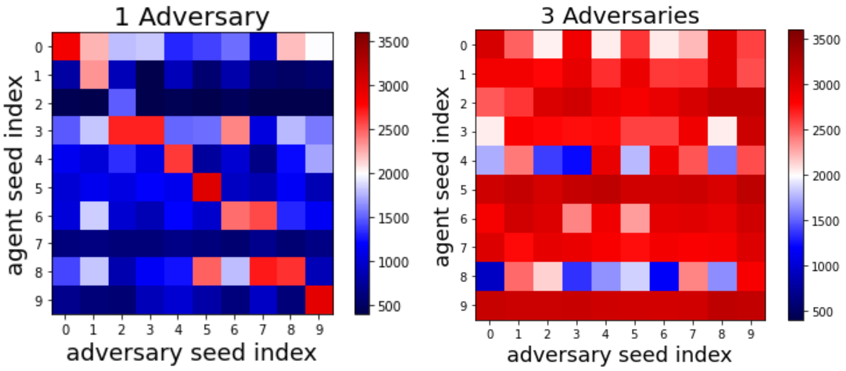


Figure 2: Average cumulative reward under swaps for one adversary training (left) and three-adversary training (right). Each square corresponds to 20 trials. In the three adversary case, each square is the average performance against the adversaries from that seed.

6 Results

H1. Analysis of Overfitting

A globally minimax optimal adversary should be unexploitable and have a lower bound on performance against any adversary in the adversary class. We investigate the optimality of our policy by asking whether the minimax agent is robust to swaps of adversaries from different training runs, i.e. different seeds. Fig. 2 shows the result of these swaps for the one adversary and three adversary case. The diagonal corresponds to playing against the adversaries the agent was trained with while every other square corresponds to playing against adversaries from a different seed. To simplify presentation, in the three adversary case, each square is the average performance against all the adversaries from that seed.

We observe that the agent trained against three adversaries is robust under swaps while the single adversary case is not. For the single adversary case, the mean performance of the agent in each seed is high against its own set of adversaries (the diagonal). This corresponds to the mean reward that would be reported at the end of training. Looking at just the reward is deceptive, the agent is still highly exploitable, as can be seen by its extremely sub-par performance against an adversary from any other seed. Since the adversaries off-diagonal are feasible adversaries, this suggests that we have found a poor local optimum of the objective.

In contrast, the three adversary case is generally robust regardless of which adversary it plays against, suggesting that the use of additional adversaries has made the agent more robust. Of course, it's possible that the adversaries are simply weaker, but as we discuss in Sec. H2., the improved performance on transfer tasks suggests that the robustness across seed swaps is indicative of a genuine improvement in robustness.

	Validation					Test				
Hopper	0 Adv	DR	1 Adv	3 Adv	5 Adv	0 Adv	DR	1 Adv	3 Adv	5 Adv
Mean Rew.	1182	2662	1094	**2039**	2021	472	1636	913	**1598**	1565
% Change		125	-7.4	**72.6**	71		246	93.4	**238**	231
	Validation					Test				
Cheetah	0 Adv	DR	1 Adv	3 Adv	5 Adv	0 Adv	DR	1 Adv	3 Adv	5 Adv
Mean Rew.	5659	3864	5593	5912	**6323**	5592	3656	5664	6046	**6406**
% Change		-32	-1.2	4.5	**11.7**		-35	1.3	8.1	**14.6**
	Validation					Test				
Ant	0 Adv	DR	1 Adv	3 Adv	5 Adv	0 Adv	DR	1 Adv	3 Adv	5 Adv
Mean Rew.	6336	6743	6349	6432	**6438**	2908	3613	3206	**3272**	3203
% Change		6.4	0.2	1.5	**1.6**		24.3	10.2	**12.5**	10.2

Table 1: Average reward and % change from vanilla PPO (0 Adv) for Hopper, Cheetah, and Ant environments across ten seeds and across the validation (left) or holdout test set (right). Across all environments, we see consistently higher robustness using RAP than the minimax adversary. Most robust adversarial approach is bolded.

H2. Adversary Population Performance

Here we present the results from the validation and holdout test sets described in Section 5.1. We compare the performance of training with adversary populations of size three and five against vanilla PPO, the domain randomization oracle, and the single minimax adversary.

Fig.3 shows the average reward (the average of ten seeds across the validation or test sets respectively) for each environment. Table 1 gives the corresponding numerical values and the percent change of each policy from the baseline. Standard deviations are omitted on the test set due to wide variation in task difficulty; the individual tests that we aggregate here are reported in the Appendix Sec. C with appropriate error bars. In all environments we achieve a higher reward across both the validation and holdout test set using RAP of size three and/or five when compared to the single minimax adversary case. These results from testing on new environments with altered dynamics supports hypothesis H1. that training with a population of adversaries leads to more robust policies than training with a single adversary.

For a more detailed comparison of robustness across the validation set, Fig. 4 shows heatmaps of the performance across all the mass, friction coefficient combinations. Here we highlight the heatmaps for Hopper and Half Cheetah for vanilla PPO, domain randomization, single adversary, and best adversary population size. Additional heatmaps for other adversary population sizes and the Ant environment can be found in Appendix Sec. C. Note that Fig. 4a is an example of a case where a single adversary has negligible effect on or slightly reduces the performance of the resultant policy on the validation set. This supports our hypothesis that a single adversary can actually lower the robustness of an agent. This result is in contrast to those observed in [18]; we conjecture that their hand-designed parametrization of the adversary (forces applied to carefully selected leg joints) may be the cause of the difference.

H3. Effect of Domain Randomization Parametrization

From Fig. 3, we see that in the Ant and Hopper domains, the oracle achieves the highest transfer reward in the validation set as expected since the oracle is trained directly on the validation set. Interestingly, we found that the domain randomization policy performed much worse on the Half Cheetah environment, despite having access to the mass and friction coefficients during training. Looking at the performance for each mass and friction combination in Fig. 4b, we found that the DR agent was able to perform much better at the low friction coefficients and learned to prioritize those values at the cost of significantly worse performance on average. This highlights a potential issue with domain randomization: while training across a wide variety of dynamics parameters can increase robustness, naive parametrizations can cause the policy to exploit subsets of the randomized domain and lead to a brittle policy.

We hypothesize that this is due to the DR objective in Eq. 4 optimizing in expectation over the sampling range. To test this, we created a separate range of ‘good’ friction parameters $[0.5, 1.5]$ and compared the robustness of a DR policy trained with a ‘good‘ range against a DR policy trained with a ‘bad’ range $[0.1, 0.9]$ in Fig. 5. Here we see that a ‘good’ parametrization leads to the expected result where domain randomization is the most robust. We observe that domain randomization, under

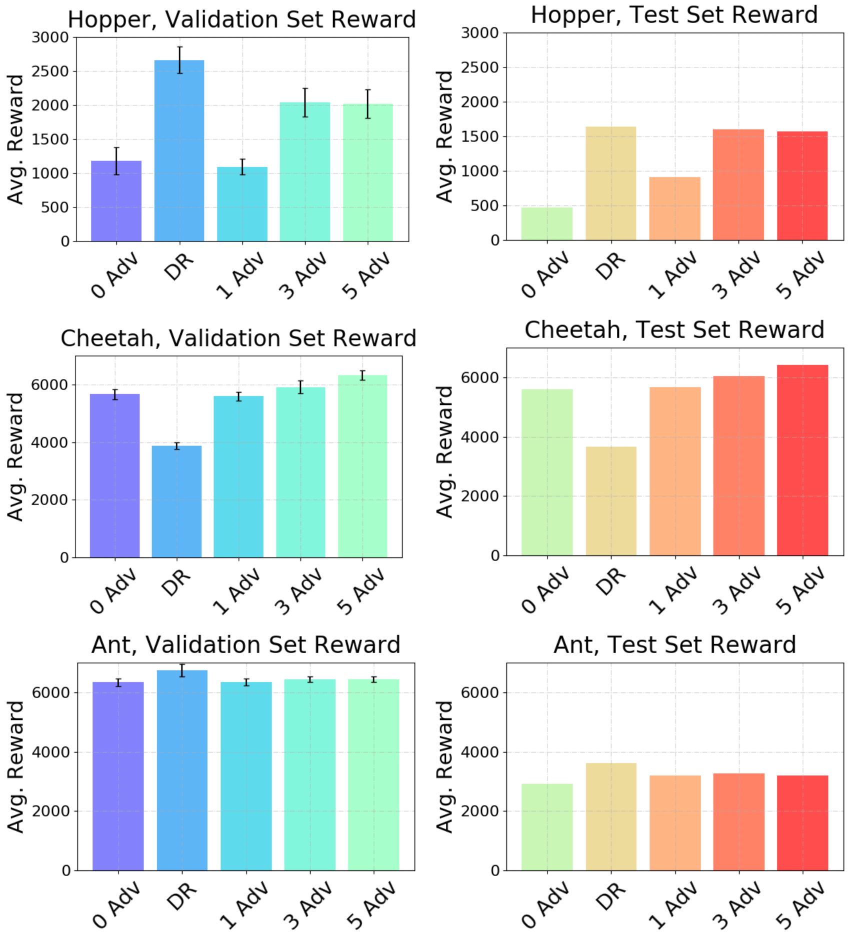


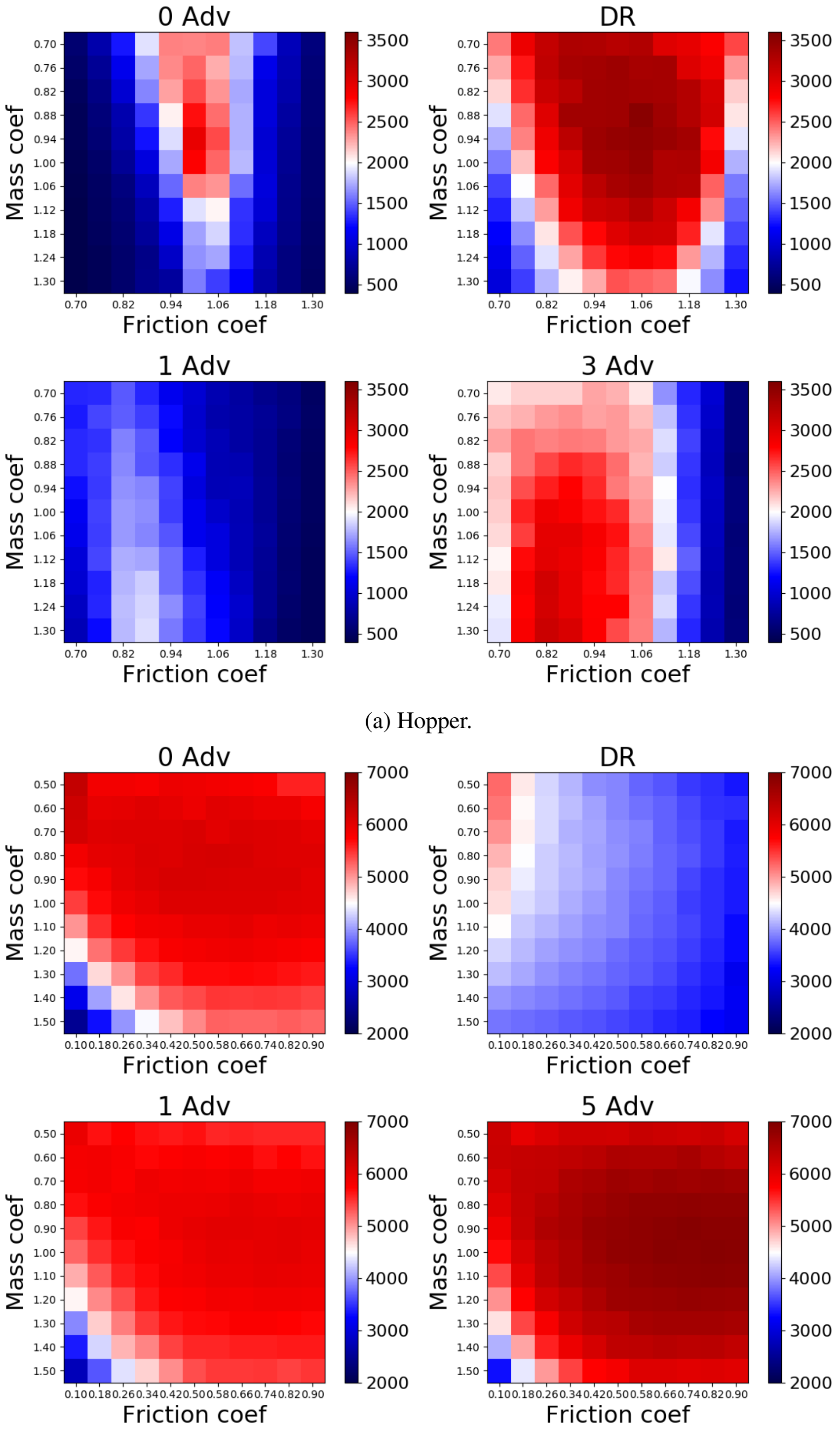
Figure 3: Average reward for Ant, Hopper, and Cheetah environments across ten seeds and across the validation set (left column) and across the holdout test set (right column). We compare vanilla PPO, the domain randomization oracle, and the minimax adversary against RAP of size three and five. Bars represent the mean and the arms represent the std. deviation. Both are computed over 20 rollouts for each test-set sample. The std. deviation for the test set are not reported here for visual clarity due to the large variation in holdout test difficulty.

the ‘bad’ paramtrization underperforms adversarial training on the validation set despite the validation set literally constituting the training set for domain randomization. This suggests that underlying optimization difficulties are partially to blame for the poor performance of domain randomization. Notably, the adversary-based methods are not susceptible to the same parametrization issues.

Prior work, EPOpt [20], has addressed this issue by replacing the uniform average across distributions with the conditional value at risk (CVaR) [4, 24], a soft version of the minimax objective where the optimization is only performed over a small percentage of the worst performing parameters. This interesting approach to align the domain randomization objective with the minimax objective could be made compatible with our approach by training using a subset of the strongest adversaries.

H4. Increasing Adversary Population Size

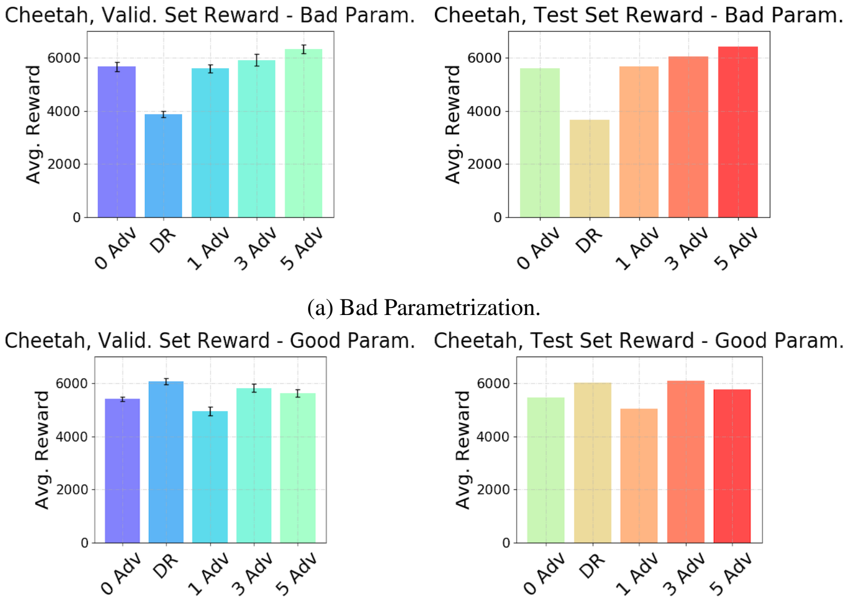
We investigate whether **RAP** is robust to adversary number as this would be a useful property to minimize hyperparameter search. Here we hypothesize that while having more adversaries can represent a wider range of dynamics to learn to be robust to, we expect there to be diminishing returns due to the decreased batch size that each adversary receives (total number of environment steps is held constant across all training variations). We expect decreasing batch size to lead to worse agent policies since the batch will contain under-trained adversary policies that the agent will learn to exploit. We cap the number of adversaries at eleven as our machines ran out of memory at this value.



(a) Hopper.

(b) Half Cheetah.

Figure 4: Average reward across ten seeds on each validation set parametrization – friction coefficient on the x-axis and mass coefficient on the y-axis. Going from left to right: Row 1 - 0 Adversary and Domain Randomization (Hopper), Row 2 - 1 Adversary and 3 Adversaries (Hopper), Row 3 - 0 Adversary and Domain Randomization (Cheetah), Row 4 - 1 Adversary and 5 Adversaries (Cheetah).



(a) Bad Parametrization.

(b) Good Parametrization.

Figure 5: Average reward for Half Cheetah environment across ten seeds. (a) shows the average reward when trained with a 'bad' friction parametrization which lead to DR not learning a robust agent policy, and (b) shows the average reward when trained with a 'good' friction parametrization.

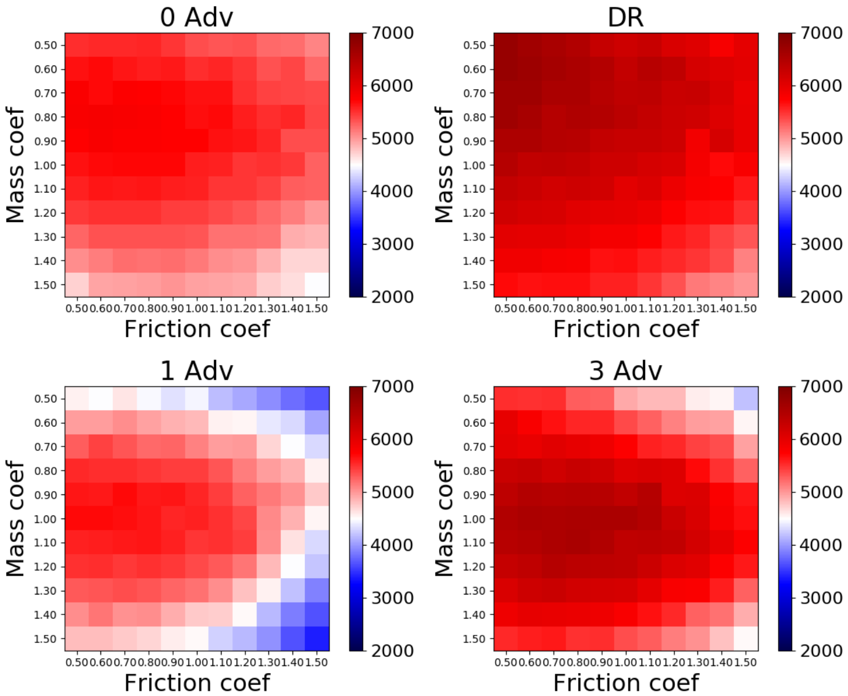


Figure 6: Half Cheetah Heatmap (Good parametrization) – friction coefficient on the x-axis and mass coefficient on the y-axis. Going from left to right: Row 1 - 0 adversary and Domain Randomization, Row 2 - 1 Adversary and 3 Adversaries.

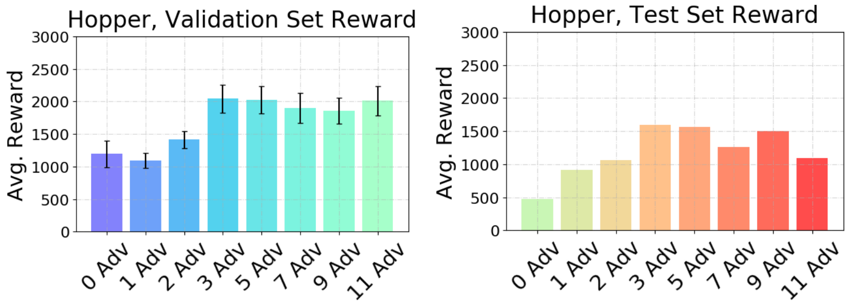


Figure 7: Average reward for Hopper across varying adversary number. Due to the wide variation in test set difficulty, standard deviations are not depicted here.

We run ten seeds for every adversary value and Fig. 7 shows the results for Hopper. Agent robustness on the test set increases monotonically up to three adversaries and roughly begins to decrease after that point. This suggests that a trade-off between adversary number and performance exists although we do not definitively show that diminishing batch sizes is the source of this trade-off. However, we observe in Fig. 3 that both three and five adversaries perform well across all studied Mujoco domains.

7 Conclusions and Future Work

In this work we demonstrate that the use of a single adversary to approximate the solution to a minimax problem does not consistently lead to improved robustness. We propose a solution through the use of multiple adversaries (RAP), and demonstrate that this provides robustness across a variety of robotics benchmarks. We also compare RAP with domain randomization and demonstrate that while DR can lead to a more robust policy, it requires careful parametrization of the domain we sample from to ensure robustness. RAP does not require this tuning, allowing for use in domains where appropriate tuning requires extensive prior knowledge or expertise.

There are several open questions stemming from this work. While we empirically demonstrate the effects of RAP, we do not have a compelling theoretical understanding of why multiple adversaries are helping. Perhaps RAP helps approximate a mixed Nash equilibrium as discussed in Sec. 1 or perhaps population based training increases the likelihood that one of the adversaries is strong? Would the benefits of RAP disappear if a single adversary had the ability to represent mixed Nash (for example, by adding a source of randomness to the adversary state)? Another interesting question to ask is whether the minimax games described here satisfy the "games-of-skill" hypothesis [5] which would provide an optimization-based reason for including adversary populations.

There are some interesting extensions of this work that we would like to pursue. We have looked at the robustness of our approach in simulated settings; future work will examine whether this robustness transfers to real-world settings. Additionally, our agents are currently memory-less and therefore cannot perform adversary identification; it would be worthwhile to see if the auxiliary task of adversary identification leads to a robust system-identification procedure that improves transfer performance. Our adversaries can also be viewed as forming a task distribution, allowing them to be used in continual learning approaches like MAML [15] where domain randomization is frequently used to construct task distributions.

Finally, here we apply adversary populations to the noisy robust MDP; applying the adversary action to the agent action represents a restricted class of dynamical systems. The transfer tests used in this work may not even be included in the set of dynamical systems that can be represented by this adversary class; this restriction may be reducing the transfer performance of the minimax approach. In future work we would like to consider a wider range of dynamical systems by using a more powerful adversary class that can control the dynamics directly.

Acknowledgments

The authors would like to thank Lerrel Pinto for help understanding and reproducing "Robust Adversarial Reinforcement Learning" as well as insightful discussions of our problem. Additionally, we would like to thank Natasha Jaques and Michael Dennis who helped us develop intuition for what the single adversary case might be flawed. Eugene Vinitsky is a recipient of an NSF Graduate Research Fellowship and funded by National Science Foundation under Grant Number CNS-1837244. Yuqing Du is funded by a Berkeley AI Research fellowship & ONR through PECASE N000141612723. Computational resources for this work were provided by an AWS Machine Learning Research grant. This material is also based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) award number CID DE-EE0008872. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] R. Antonova, S. Cruciani, C. Smith, and D. Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[3] N. Brown and T. Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[4] Y. Chow, A. Tamar, S. Mannor, and M. Pavone. Risk-sensitive and robust decision-making: a cvar optimization approach. In *Advances in Neural Information Processing Systems*, pages 1522–1530, 2015.

[5] W. M. Czarnecki, G. Gidel, B. Tracey, K. Tuyls, S. Omidshafiei, D. Balduzzi, and M. Jaderberg. Real world games look like spinning tops. *arXiv preprint arXiv:2004.09468*, 2020.

[6] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell. Adversarial policies: Attacking deep reinforcement learning. *arXiv preprint arXiv:1905.10615*, 2019.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[8] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

[9] P. Kamalaruban, Y.-T. Huang, Y.-P. Hsieh, P. Rolland, C. Shi, and V. Cevher. Robust reinforcement learning via adversarial training with langevin dynamics. *arXiv preprint arXiv:2002.06063*, 2020.

[10] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[11] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica. Rllib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381*, 2017.

[12] S. H. Lim, H. Xu, and S. Mannor. Reinforcement learning in robust markov decision processes. In *Advances in Neural Information Processing Systems*, pages 701–709, 2013.

[13] Y. Liu, P. Ramachandran, Q. Liu, and J. Peng. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399*, 2017.

[14] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull. Active domain randomization. *arXiv preprint arXiv:1904.04762*, 2019.

[15] A. Nagabandi, C. Finn, and S. Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018.

[16] A. Nilim and L. El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[17] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

[18] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

[19] M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[20] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

[21] F. Sadeghi and S. Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

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

[23] L. Shapley. Some topics in two-person games. *Advances in game theory*, 52:1–29, 1964.

[24] A. Tamar, Y. Glassner, and S. Mannor. Optimizing the cvar via sampling. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[25] A. Tamar, S. Mannor, and H. Xu. Scaling up robust mdps using function approximation. In *International Conference on Machine Learning*, pages 181–189, 2014.

[26] C. Tessler, Y. Efroni, and S. Mannor. Action robust reinforcement learning and applications in continuous control. *arXiv preprint arXiv:1901.09184*, 2019.

[27] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

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

[29] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[30] K. Zhou and J. C. Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.

Appendices

A Full Description of the MDPs

We use the Mujoco ant, cheetah, and hopper environments as a test of the efficacy of our strategy versus the 0 adversary, 1 adversary, and domain randomization baselines. We use the Noisy Action Robust MDP formulation [26] for our adversary parametrization. If the normal system dynamics are

$$s_{k+1} = s_k + f(s_k, a_k)\Delta t$$

the system dynamics under the adversary are

$$s_{k+1} = s_k + f(s_k, a_k + a_k^{\text{adv}})\Delta t$$

where a_k^{adv} is the adversary action at time k.

The notion here is that the adversary action is passed through the dynamics function and represents some additional set of dynamics. It is standard to clip actions within some boundary but clipping the sum would allow the agent to "cancel" the adversary by always keeping its action at the bounds of the action space. Since we want the adversary to always affect the dynamics irrespective of agent action, we clip the agent and adversary actions separately. The agent is clipped between $[-1, 1]$ in all environments and the adversary is clipped between $[-.25, .25]$.

The MDP through which we train the agent policy is characterized by the following states, actions, and rewards:

- $s_t^{\text{agent}} = [o_t, a_{t-1}]$ where o_t is an observation returned by the environment, and a_t is the action taken by the agent.
- We use the standard rewards provided by the OpenAI Gym Mujoco environments at `https://github.com/openai/gym/tree/master/gym/envs/mujoco`. For the exact functions, please refer to the code at `https://github.com/eugenevinitsky/robust_RL_multi_adversary`.
- $a_t^{\text{agent}} \in [a_{\text{min}}, a_{\text{max}}]^n$.

The MDP for adversary i is the following:

- $s_t = s_t^{\text{agent}}$. The adversary sees the same states as the agent.
- The adversary reward is the negative of the agent reward.
- $a_t^{\text{adv}} \in \left[a_{\text{min}}^{\text{adv}}, a_{\text{max}}^{\text{adv}}\right]^n$.

For our domain randomization Hopper baseline, we use the following randomization: at each rollout, we scale the friction of all joints by a single value uniformly sampled from [0.7, 1.3]. We also randomly scale the mass of the 'torso' link by a single value sampled from [0.7, 1.3]. For Half-Cheetah and Ant the range for friction is [0.1, 0.9] and for mass the range is [0.5, 1.5].

B Holdout Tests

In this section we describe in detail all of the holdout tests used.

B.1 Hopper

The Mujoco geom properties that we modified are attached to a particular body and determine its appearance and collision properties. For the Mujoco holdout transfer tests we pick a subset of the hopper 'geom' elements and scale the contact friction values by maximum friction coefficient, 1.3. Likewise, for the rest of the 'geom' elements, we scale the contact friction by the minimum value of 0.7. The body geoms and their names are visible in Fig. 8.

The exact combinations and the corresponding test name are indicated in Table 2 for Hopper.

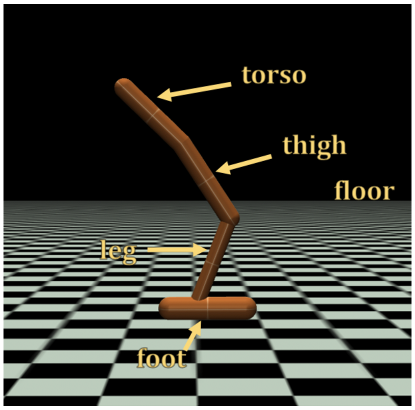


Figure 8: Labelled Body Segments of Hopper

Table 2: Hopper Holdout Test Descriptions

Test	Body with Friction Coeff 1.3	Body with Friction Coeff 0.7
A	Torso, Leg	Floor, Thigh, Foot
B	Floor, Thigh	Torso, Leg, Foot
C	Foot, Leg	Floor, Torso, Thigh
D	Torso, Thigh, Floor	Foot, Leg
E	Torso, Foot	Floor, Thigh, Leg
F	Floor, Thigh, Leg	Torso, Foot
G	Floor, Foot	Torso, Thigh, Leg
H	Thigh, Leg	Floor, Torso, Foot

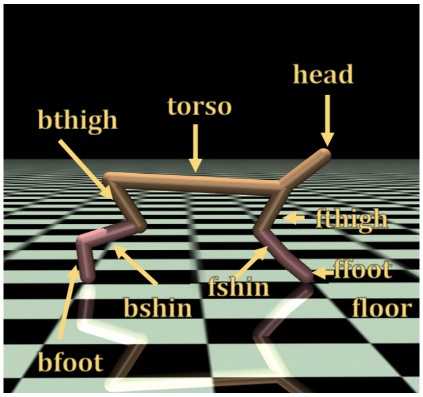


Figure 9: Labelled Body Segments of Cheetah

Table 3: Cheetah Holdout Test Descriptions. Joints in the table receive the maximum friction coefficient of 0.9. Joints not indicated have friction coefficient 0.1

Test	Geom with Friction Coeff 0.9
A	Torso, Head, Fthigh
B	Floor, Head, Fshin
C	Bthigh, Bshin, Bfoot
D	Floor, Torso, Head
E	Floor, Bshin, Ffoot
F	Bthigh, Bfoot, Ffoot
G	Bthigh, Fthigh, Fshin
H	Head, Fshin, Ffoot

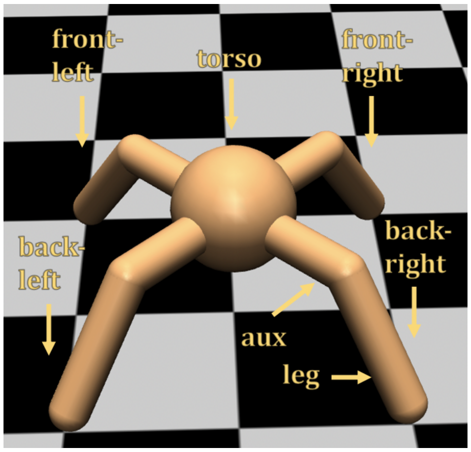


Figure 10: Labelled Body Segments of Ant

Table 4: Ant Holdout Test Descriptions. Joints in the table receive the maximum friction coefficient of 0.9. Joints not indicated have friction coefficient 0.1

Test	Geom with Friction Coeff 0.9
A	Front-Leg-Left, Aux-Front-Left, Aux-Back-Left
B	Torso, Aux-Front-Left, Back-Leg-Right
C	Front-Leg-Right, Aux-Front-Right, Back-Leg-Left
D	Torso, Front-Leg-Left, Aux-Front-Left
E	Front-Leg-Left, Aux-Front-Right, Aux-Back-Right
F	Front-Leg-Right, Back-Leg-Left, Aux-Back-Right
G	Front-Leg-Left, Aux-Back-Left, Back-Leg-Right
H	Aux-Front-Left, Back-Leg-Right, Aux-Back-Right

B.2 Cheetah

The Mujoco geom properties that we modified are attached to a particular body and determine its appearance and collision properties. For the Mujoco holdout transfer tests we pick a subset of the cheetah ‘geom’ elements and scale the contact friction values by maximum friction coefficient, 0.9. Likewise, for the rest of the ‘geom’ elements, we scale the contact friction by the minimum value of 0.1. The body geoms and their names are visible in Fig. 9.

The exact combinations and the corresponding test name are indicated in Table 3 for Hopper.

B.3 Ant

We will use torso to indicate the head piece, leg to refer to one of the four legs that contact the ground, and ’aux’ to indicate the geom that connects the leg to the torso. Since the ant is symmetric we adopt a convention that two of the legs are front-left and front-right and two legs are back-left and back-right. Fig. 10 depicts the convention. For the Mujoco holdout transfer tests we pick a subset of the ant ‘geom’ elements and scale the contact friction values by maximum friction coefficient, 0.9. Likewise, for the rest of the ‘geom’ elements, we scale the contact friction by the minimum value of 0.1.

The exact combinations and the corresponding test name are indicated in Table 4 for Hopper.

C Results

Here we recompute the values of all the results and display them with appropriate standard deviations in tabular form. Tables 5, 6, 7 contain the test results with appropriate standard deviations for Hopper, Half-Cheetah, and Ant respectively.

Test Name	0 Adv	1 Adv	3 Adv	Five Adv	Domain Rand
Test A	410 ± 140	1170 ± 570	**2210 ± 630**	2090 ± 920	1610 ± 310
Test B	430 ± 150	1160 ± 540	**2240 ± 730**	2200 ± 880	1610 ± 290
Test C	560 ± 120	490 ± 150	610 ± 250	580 ± 120	**1660 ± 260**
Test D	420 ± 150	1140 ± 560	**2220 ± 680**	2130 ± 890	1612 ± 360
Test E	550 ± 120	500 ± 150	600 ± 240	590 ± 120	**1680 ± 280**
Test F	420 ± 150	1200 ± 620	2080 ± 750	**2160 ± 890**	1650 ± 360
Test H	560 ± 130	500 ± 140	600 ± 230	600 ± 140	**1710 ± 370**
Test G	420 ± 150	1160 ± 590	**2210 ± 680**	2160 ± 920	1560 ± 340

Table 5: Results on holdout tests for each of the tested approaches for Hopper. Bolded values have the highest mean

Test Name	0 Adv	1 Adv	3 Adv	Five Adv	Domain Rand
Test A	4400 ± 2160	5110 ± 730	4960 ± 1280	**5560 ± 1060**	2800 ± 1540
Test B	6020 ± 880	5980 ± 290	6440 ± 1620	**6880 ± 1090**	3340 ± 600
Test C	5880 ± 1030	5730 ± 640	**6740 ± 1190**	6410 ± 790	4280 ± 240
Test D	5990 ± 940	5960 ± 260	6430 ± 1610	**6880 ± 1090**	3360 ± 570
Test E	5570 ± 570	5670 ± 290	5800 ± 1316	**6530 ± 1250**	3720 ± 540
Test F	5870 ± 750	5800 ± 350	6500 ± 1100	**6770 ± 1070**	3810 ± 330
Test H	5310 ± 1060	5270 ± 700	5610 ± 720	**5660 ± 980**	4560 ± 560
Test G	5710 ± 650	5790 ± 300	5890 ± 1240	**6560 ± 1240**	3380 ± 720

Table 6: Results on holdout tests for each of the tested approaches for Half Cheetah. Bolded values have the highest mean

Test Name	0 Adv	1 Adv	3 Adv	Five Adv	Domain Rand
Test A	590 ± 650	730 ± 630	600 ± 440	560 ± 580	**900 ± 580**
Test B	5240 ± 280	5530 ± 200	5770 ± 100	5710 ± 180	**6150 ± 180**
Test C	750 ± 820	1090 ± 660	1160 ± 540	1040 ± 760	**1370 ± 800**
Test D	5220 ± 300	5560 ± 220	5770 ± 90	5660 ± 190	**6120 ± 180**
Test E	5270 ± 290	5570 ± 210	5770 ± 100	5660 ± 220	**6140 ± 150**
Test F	780 ± 860	1160 ± 570	1120 ± 580	1140 ± 870	**1390 ± 750**
Test H	130 ± 290	420 ± 300	210 ± 220	160 ± 270	**700 ± 560**
Test G	5290 ± 280	5560 ± 220	5770 ± 100	5700 ± 190	**6150 ± 160**

Table 7: Results on holdout tests for each of the tested approaches for Ant. Bolded values have the highest mean

Finally, we place the heatmaps for Ant here for reference in Fig. 11.

D Cost and Hyperparameters

Here we reproduce the hyperparameters we used in each experiment and compute the expected runtime and cost of each experiment. Numbers indicated in $\{\}$ were each used for one run. Otherwise the parameter was kept fixed at the indicated value.

D.1 Hyperparameters

For Mujoco the hyperparameters are:

- Learning rate:
  - $\{.0003, .0005\}$ for half cheetah
  - $\{.0005, .00005\}$ for hopper
- : Bounds on adversary action space: $[-0.25, 0.25]$
- Generalized Advantage Estimation λ
  - $\{0.9, 0.95, 1.0\}$ for half cheetah

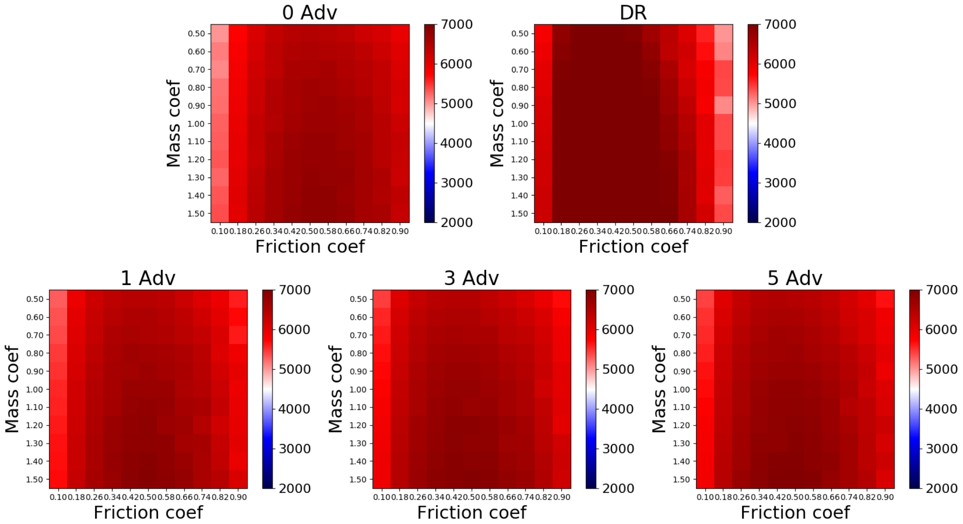


Figure 11: Ant Heatmap: Average reward across 10 seeds on each validation set (mass, friction) parametrization.

  - $\{0.5, 0.9, 1.0\}$ for hopper and ant
- Discount factor $\gamma = 0.995$
- Training batch size: 100000
- SGD minibatch size: 640
- Number of SGD steps per iteration: 10
- Number of iterations: 700
- We set the seed to 0 for all hyperparameter runs.
- The maximum horizon is 1000 steps.

For the validation across seeds we used 10 seeds ranging from 0 to 9. Values of hyperparameters selected for each adversary number can be found by consulting the code-base. All other hyperparameters are the default values in RLlib [11] 0.8.0.

D.2 Cost

For all of our experiments we used AWS EC2 c4.8xlarge instances which come with 36 virtual CPUs. For the Mujoco experiments, we use 2 nodes and 11 CPUs per hyper-parameter, leading to one full hyper-parameter sweep fitting onto the 72 CPUs. We run the following set of experiments and ablations, each of which takes 8 hours.

- 0 adversaries
- 1 adversary
- 3 adversaries
- 5 adversaries
- Domain randomization

for a total of 5 experiments for each of Hopper, Cheetah, Ant. For the best hyperparameters and each experiment listed above we run a seed search with 6 CPUs used per-seed, a process which takes about 12 hours. This leads to a total of $2 * 8 * 5 * 3 + 2 * 12 * 3 * 5 = 600$ node hours and $36 * 600 \approx 22000$ CPU hours. At a cost of ≈ 0.3 dollars per node per hour for EC2 spot instances, this gives ≈ 180 dollars to fully reproduce our results for this experiment. If the chosen hyperparameters are used and only the seeds are swept, this is ≈ 100 dollars.