Robust Adversarial Reinforcement Learning

Lerrel Pinto [1]   **James Davidson** [2]   **Rahul Sukthankar** [3]   **Abhinav Gupta** [1][3]

Abstract

Deep neural networks coupled with fast simulation and improved computation have led to recent successes in the field of reinforcement learning (RL). However, most current RL-based approaches fail to generalize since: (a) the gap between simulation and real world is so large that policy-learning approaches fail to transfer; (b) even if policy learning is done in real world, the data scarcity leads to failed generalization from training to test scenarios (e.g., due to different friction or object masses). Inspired from H_∞ control methods, we note that both modeling errors and differences in training and test scenarios can be viewed as extra forces/disturbances in the system. This paper proposes the idea of robust adversarial reinforcement learning (RARL), where we train an agent to operate in the presence of a destabilizing adversary that applies disturbance forces to the system. The jointly trained adversary is reinforced – that is, it learns an optimal destabilization policy. We formulate the policy learning as a zero-sum, minimax objective function. Extensive experiments in multiple environments (InvertedPendulum, HalfCheetah, Swimmer, Hopper, Walker2d and Ant) conclusively demonstrate that our method (a) improves training stability; (b) is robust to differences in training/test conditions; and c) outperform the baseline even in the absence of the adversary.

1. Introduction

High-capacity function approximators such as deep neural networks have led to increased success in the field of reinforcement learning (Mnih et al., 2015; Silver et al., 2016; Gu et al., 2016; Lillicrap et al., 2015; Mordatch et al., 2015). However, a major bottleneck for such

[1]Carnegie Mellon University [2]Google Brain [3]Google Research. Correspondence to: Lerrel Pinto <lerrelp@cs.cmu.edu>.

policy-learning methods is their reliance on data: training high-capacity models requires huge amounts of training data/trajectories. While this training data can be easily obtained for tasks like games (e.g., Doom, Montezuma's Revenge) (Mnih et al., 2015), data-collection and policy learning for real-world physical tasks are significantly more challenging.

There are two possible ways to perform policy learning for real-world physical tasks:

- **Real-world Policy Learning:** The first approach is to learn the agent's policy in the real-world. However, training in the real-world is too expensive, dangerous and time-intensive leading to scarcity of data. Due to scarcity of data, training is often restricted to a limited set of training scenarios, causing overfitting. If the test scenario is different (e.g., different friction coefficient), the learned policy fails to generalize. Therefore, we need a learned policy that is robust and generalizes well across a range of scenarios.
- **Learning in simulation:** One way of escaping the data scarcity in the real-world is to transfer a policy learned in a simulator to the real world. However the environment and physics of the simulator are not exactly the same as the real world. This reality gap often results in unsuccessful transfer if the learned policy isn't robust to modeling errors (Christiano et al., 2016; Rusu et al., 2016).

Both the test-generalization and simulation-transfer issues are further exacerbated by the fact that many policy-learning algorithms are stochastic in nature. For many hard physical tasks such as Walker2D (Erez et al., 2011), only a small fraction of runs leads to stable walking policies. This makes these approaches even more time and data-intensive. What we need is an approach that is significantly more stable/robust in learning policies across different runs and initializations while requiring less data during training.

So, how can we model uncertainties and learn a policy robust to all uncertainties? How can we model the gap between simulations and real-world? We begin with the insight that modeling errors can be viewed as extra forces/disturbances in the system (Başar & Bernhard, 2008). For example, high friction at test time might be modeled as extra forces at contact points against the di-

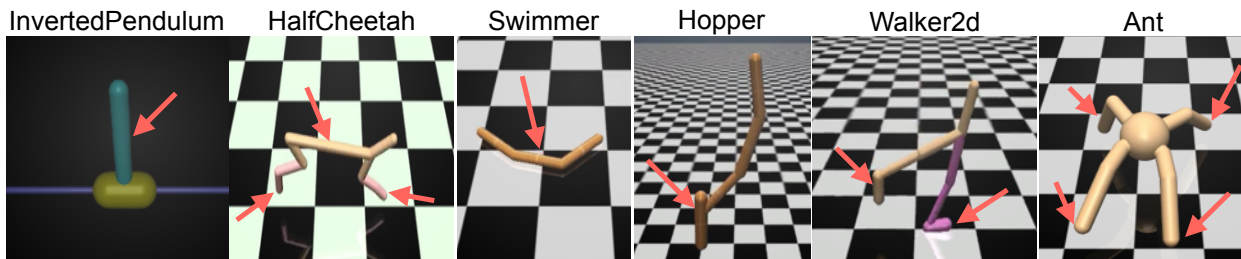| InvertedPendulum | HalfCheetah | Swimmer | Hopper | Walker2d | Ant |

Figure 1. We evaluate RARL on a variety of OpenAI gym problems. The adversary learns to apply destabilizing forces on specific points (denoted by red arrows) on the system, encouraging the protagonist to learn a robust control policy. These policies also transfer better to new test environments, with different environmental conditions and where the adversary may or may not be present.

rection of motion. Inspired by this observation, this paper proposes the idea of modeling uncertainties via an adversarial agent that applies disturbance forces to the system. Moreover, the adversary is reinforced – that is, it learns an optimal policy to thwart the original agent's goal. Our proposed method, Robust Adversarial Reinforcement Learning (RARL), jointly trains a pair of agents, a protagonist and an adversary, where the protagonist learns to fulfil the original task goals while being robust to the disruptions generated by its adversary.

We perform extensive experiments to evaluate RARL on multiple OpenAI gym environments like InvertedPendulum, HalfCheetah, Swimmer, Hopper, Walker2d and Ant (see Figure 1). We demonstrate that our proposed approach is: **(a) Robust to model initializations:** The learned policy performs better given different model parameter initializations and random seeds. This alleviates the data scarcity issue by reducing sensitivity of learning. **(b) Robust to modeling errors and uncertainties:** The learned policy generalizes significantly better to different test environment settings (e.g., with different mass and friction values).

1.1. Overview of RARL

Our goal is to learn a policy that is robust to modeling errors in simulation or mismatch between training and test scenarios. For example, we would like to learn policy for Walker2D that works not only on carpet (training scenario) but also generalizes to walking on ice (test scenario). Similarly, other parameters such as the mass of the walker might vary during training and test. One possibility is to list all such parameters (mass, friction etc.) and learn an ensemble of policies for different possible variations (Rajeswaran et al., 2016). But explicit consideration of all possible parameters of how simulation and real world might differ or what parameters can change between training/test is infeasible.

Our core idea is to model the differences during training and test scenarios via extra forces/disturbances in the system. Our hypothesis is that if we can learn a policy that is robust to all disturbances, then this policy will be robust to changes in training/test situations; and hence generalize well. But is it possible to sample trajectories under all possible disturbances? In unconstrained scenarios, the space of possible disturbances could be larger than the space of possible actions, which makes sampled trajectories even sparser in the joint space.

To overcome this problem, we advocate a two-pronged approach:

(a) Adversarial agents for modeling disturbances: Instead of sampling all possible disturbances, we jointly train a second agent (termed the adversary), whose goal is to impede the original agent (termed the protagonist) by applying destabilizing forces. The adversary is rewarded only for the failure of the protagonist. Therefore, the adversary learns to sample hard examples: disturbances which will make original agent fail; the protagonist learns a policy that is robust to any disturbances created by the adversary.

(b) Adversaries that incorporate domain knowledge: The naive way of developing an adversary would be to simply give it the same action space as the protagonist – like a driving student and driving instructor fighting for control of a dual-control car. However, our proposed approach is much richer and is not limited to symmetric action spaces – we can exploit domain knowledge to: focus the adversary on the protagonist's weak points; and since the adversary is in a simulated environment, we can give the adversary "super-powers" – the ability to affect the robot or environment in ways the protagonist cannot (e.g., suddenly change a physical parameter like frictional coefficient or mass).

2. Background

Before we delve into the details of RARL, we first outline our terminology, standard reinforcement learning setting and two-player zero-sum games from which our paper is inspired.

2.1. Standard reinforcement learning on MDPs

In this paper we examine continuous space MDPs that are represented by the tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, s_0)$, where $\mathcal{S}$ is a set of continuous states and $\mathcal{A}$ is a set of continuous actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the transition probability, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, γ is the discount factor, and s_0 is the initial state distribution.

Batch policy algorithms like REINFORCE (Williams, 1992), NPG (Kakade, 2002) and TRPO (Schulman et al., 2015) attempt to learn a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that maximizes the cumulative discounted reward $\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$. Here, θ denotes the parameters for the policy π which takes action a_t given state s_t at timestep t.

2.2. Two-player zero-sum discounted games

The adversarial setting we propose can be expressed as a two player γ discounted zero-sum Markov game (Littman, 1994; Perolat et al., 2015). This game MDP can be expressed as the tuple: $(\mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{P}, r, \gamma, s_0)$ where $\mathcal{A}_1$ and $\mathcal{A}_2$ are the continuous set of actions the players can take. $\mathcal{P} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{S} \rightarrow \mathbb{R}$ is the transition probability density and $r : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \rightarrow \mathbb{R}$ is the reward of both players. If player 1 (protagonist) is playing strategy μ and player 2 (adversary) is playing the strategy ν, the reward function is $r_{\mu,\nu} = E_{a^1 \sim \mu(.|s), a^2 \sim \nu(.|s)}[r(s, a^1, a^2)]$. A zero-sum two-player game can be seen as player 1 maximizing the γ discounted reward while player 2 is minimizing it.

3. Robust Adversarial RL

3.1. Robust Control via Adversarial Agents

Our goal is to learn the policy of the protagonist (denoted by μ) such that it is better (higher reward) and robust (generalizes better to variations in test settings). In the standard reinforcement learning setting, for a given transition function $\mathcal{P}$, we can learn policy parameters θ^μ such that the expected reward is maximized where expected reward for policy μ from the start s_0 is

$$\rho(\mu; \theta^\mu, \mathcal{P}) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | s_0, \mu, \mathcal{P}\right]. \quad (1)$$

Note that in this formulation the expected reward is conditioned on the transition function since the the transition function defines the roll-out of states. In standard-RL settings, the transition function is fixed (since the physics engine and parameters such as mass, friction are fixed). However, in our setting, we assume that the transition function will have modeling errors and that there will be differences between training and test conditions. Therefore, in our general setting, we should estimate policy parameters θ^μ such that we maximize the expected reward over different possi-

ble transition functions as well. Therefore,

$$\rho(\mu; \theta^\mu) = \mathbb{E}_{\mathcal{P}}\left[\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | s_0, \mu, \mathcal{P}\right]\right]. \quad (2)$$

Optimizing for the expected reward over *all* transition functions optimizes *mean performance*, which is a risk neutral formulation that assumes a known distribution over model parameters. A large fraction of policies learned under such a formulation are likely to fail in a different environment. Instead, inspired by work in robust control (Tamar et al., 2014; Rajeswaran et al., 2016), we choose to optimize for conditional value at risk (CVaR):

$$\rho_{RC} = \mathbb{E}\left[\rho | \rho \leq \mathcal{Q}_\alpha(\rho)\right] \quad (3)$$

where $\mathcal{Q}_\alpha(\rho)$ is the α-quantile of ρ-values. Intuitively, in robust control, we want to maximize the worst-possible ρ-values. But how do you tractably sample trajectories that are in worst α-percentile? Approaches like EP-Opt (Rajeswaran et al., 2016) sample these worst percentile trajectories by changing parameters such as friction, mass of objects, etc. during rollouts.

Instead, we introduce an adversarial agent that applies forces on pre-defined locations, and this agent tries to change the trajectories such that reward of the protagonist is minimized. Note that since the adversary tries to minimize the protagonist's reward, it ends up sampling trajectories from worst-percentile leading to robust control-learning for the protagonist. If the adversary is kept fixed, the protagonist could learn to overfit to its adversarial actions. Therefore, instead of using either a random or a fixed-adversary, we advocate generating the adversarial actions using a learned policy ν. We would also like to point out the connection between our proposed approach and the practice of hard-example mining (Sung & Poggio, 1994; Shrivastava et al., 2016). The adversary in RARL learns to sample hard-examples (worst-trajectories) for the protagonist to learn. Finally, instead of using α as percentile-parameter, RARL is parameterized by the magnitude of force available to the adversary. As the adversary becomes stronger, RARL optimizes for lower percentiles. However, very high magnitude forces lead to very biased sampling and make the learning unstable. In the extreme case, an unreasonably strong adversary can always prevent the protagonist from achieving the task. Analogously, the traditional RL baseline is equivalent to training with an impotent (zero strength) adversary.

3.2. Formulating Adversarial Reinforcement Learning

In our adversarial game, at every timestep t both players observe the state s_t and take actions $a_t^1 \sim \mu(s_t)$ and $a_t^2 \sim \nu(s_t)$. The state transitions $s_{t+1} = \mathcal{P}(s_t, a_t^1, a_t^2)$

and a reward $r_t = r(s_t, a_t^1, a_t^2)$ is obtained from the environment. In our zero-sum game, the protagonist gets a reward $r_t^1 = r_t$ while the adversary gets a reward $r_t^2 = -r_t$. Hence each step of this MDP can be represented as $(s_t, a_t^1, a_t^2, r_t^1, r_t^2, s_{t+1})$.

The protagonist seeks to maximize the following reward function,

$$R^1 = E_{s_0 \sim \rho, a^1 \sim \mu(s), a^2 \sim \nu(s)}\left[\sum_{t=0}^{T-1} r^1(s, a^1, a^2)\right]. \quad (4)$$

Since, the policies μ and ν are the only learnable components, $R^1 \equiv R^1(\mu, \nu)$. Similarly the adversary attempts to maximize its own reward: $R^2 \equiv R^2(\mu, \nu) = -R^1(\mu, \nu)$. One way to solve this MDP game is by discretizing the continuous state and action spaces and using dynamic programming to solve. Perolat et al. (2015) and Patek (1997) show that notions of minimax equilibrium and Nash equilibrium are equivalent for this game with optimal equilibrium reward:

$$R^{1*} = \min_\nu \max_\mu R^1(\mu, \nu) = \max_\mu \min_\nu R^1(\mu, \nu) \quad (5)$$

However solutions to finding the Nash equilibria strategies often involve greedily solving N minimax equilibria for a zero-sum matrix game, with N equal to the number of observed datapoints. The complexity of this greedy solution is exponential in the cardinality of the action spaces, which makes it prohibitive (Perolat et al., 2015).

Most Markov Game approaches require solving for the equilibrium solution for a multiplayer value or minimax-Q function at each iteration. This requires evaluating a typically intractable minimax optimization problem. Instead, we focus on learning stationary policies μ^* and ν^* such that $R^1(\mu^*, \nu^*) \to R^{1*}$. This way we can avoid this costly optimization at each iteration as we just need to approximate the advantage function and not determine the equilibrium solution at each iteration.

3.3. Proposed Method: RARL

Our algorithm (RARL) optimizes both of the agents using the following alternating procedure. In the first phase, we learn the protagonist's policy while holding the adversary's policy fixed. Next, the protagonist's policy is held constant and the adversary's policy is learned. This sequence is repeated until convergence.

Algorithm 1 outlines our approach in detail. The initial parameters for both players' policies are sampled from a random distribution. In each of the N_{iter} iterations, we carry out a two-step (alternating) optimization procedure. First, for N_μ iterations, the parameters of the adversary θ^ν are held constant while the parameters θ^μ of the protagonist

---

Algorithm 1 RARL (proposed algorithm)

Input: Environment $\mathcal{E}$; Stochastic policies μ and ν
Initialize: Learnable parameters θ_0^μ for μ and θ_0^ν for ν
for i=1,2,..N_{iter} **do**
  $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$
  for j=1,2,..N_μ **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_{i-1}^\nu}, N_{\text{traj}})$
    $\theta_i^\mu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \mu, \theta_i^\mu)$
  end for
  $\theta_i^\nu \leftarrow \theta_{i-1}^\nu$
  for j=1,2,..N_ν **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_i^\nu}, N_{\text{traj}})$
    $\theta_i^\nu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{2i}, r_t^{2i})\}, \nu, \theta_i^\nu)$
  end for
end for
Return: $\theta_{N_{\text{iter}}}^\mu, \theta_{N_{\text{iter}}}^\nu$

---

are optimized to maximize R^1 (Equation 4). The *roll* function samples N_{traj} trajectories given the environment definition $\mathcal{E}$ and the policies for both the players. Note that $\mathcal{E}$ contains the transition function $\mathcal{P}$ and the reward functions r^1 and r^2 to generate the trajectories. The tth element of the ith trajectory is of the form $(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})$. These trajectories are then split such that the tth element of the ith trajectory is of the form $(s_t^i, a_t^i = a_t^{1i}, r_t^i = r_t^{1i})$. The protagonist's parameters θ^μ are then optimized using a policy optimizer. For the second step, player 1's parameters θ^μ are held constant for the next N_ν iterations. N_{traj} Trajectories are sampled and split into trajectories such that t^{th} element of the ith trajectory is of the form $(s_t^i, a_t^i = a_t^{2i}, r_t^i = r_t^{2i})$. Player 2's parameters θ^ν are then optimized. This alternating procedure is repeated for N_{iter} iterations.

4. Experimental Evaluation

We now demonstrate the robustness of the RARL algorithm: (a) for training with different initializations; (b) for testing with different conditions; (c) for adversarial disturbances in the testing environment. But first we will describe our implementation and test setting followed by evaluations and results of our algorithm.

4.1. Implementation

Our implementation of the adversarial environments build on OpenAI Gym's (Brockman et al., 2016) control environments with the MuJoCo (Todorov et al., 2012) physics simulator. Details of the environments and their corresponding adversarial disturbances are (also see Figure 1):

InvertedPendulum: The inverted pendulum is mounted on a pivot point on a cart, with the cart restricted to linear movement in a plane. The state space is 4D: position

and velocity for both the cart and the pendulum. The protagonist can apply 1D forces to keep the pendulum upright. The adversary applies a 2D force on the center of pendulum in order to destabilize it.

HalfCheetah: The half-cheetah is a planar biped robot with 8 rigid links, including two legs and a torso, along with 6 actuated joints. The 17D state space includes joint angles and joint velocities. The adversary applies a 6D action with 2D forces on the torso and both feet in order to destabilize it.

Swimmer: The swimmer is a planar robot with 3 links and 2 actuated joints in a viscous container, with the goal of moving forward. The 8D state space includes joint angles and joint velocities. The adversary applies a 3D force to the center of the swimmer.

Hopper: The hopper is a planar monopod robot with 4 rigid links, corresponding to the torso, upper leg, lower leg, and foot, along with 3 actuated joints. The 11D state space includes joint angles and joint velocities. The adversary applies a 2D force on the foot.

Walker2D: The walker is a planar biped robot consisting of 7 links, corresponding to two legs and a torso, along with 6 actuated joints. The 17D state space includes joint angles and joint velocities. The adversary applies a 4D action with 2D forces on both the feet.

Ant: The ant is a quadruped robot with a torso and 4 legs that contain 8 joints. The 111D observation space includes joint angles, joint velocities and contact forces. The adversary applies a 8D action with 2D forces on each foot.

Our implementation of RARL is built on top of rllab (Duan et al., 2016) and uses Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) as the policy optimizer. For all the tasks and for both the protagonist and adversary, we use a policy network with two hidden layers with 64 neurons each. We train both RARL and the baseline for 100 iterations on InvertedPendulum and for 500 iterations on the other tasks. Hyperparameters of TRPO are selected by grid search.

4.2. Evaluating Learned Policies

We evaluate the robustness of our RARL approach compared to the strong TRPO baseline. Since our policies are stochastic in nature and the starting state is also drawn from a distribution, we learn 50 policies for each task with different seeds/initializations. First, we report the mean and variance of cumulative reward (over 50 policies) as a function of the training iterations. Figure 2 shows the mean and variance of the rewards of learned policies for the task of HalfCheetah, Swimmer, Hopper and Walker2D. We omit the graph for InvertedPendulum because the task is easy

and both TRPO and RARL show similar performance and similar rewards. As we can see from the figure, for all the four tasks RARL learns a better policy in terms of mean reward and variance as well. This clearly shows that the policy learned by RARL is better than the policy learned by TRPO even when there is no disturbance or change of settings between training and test conditions. Table 1 reports the average rewards with their standard deviations for the best learned policy.
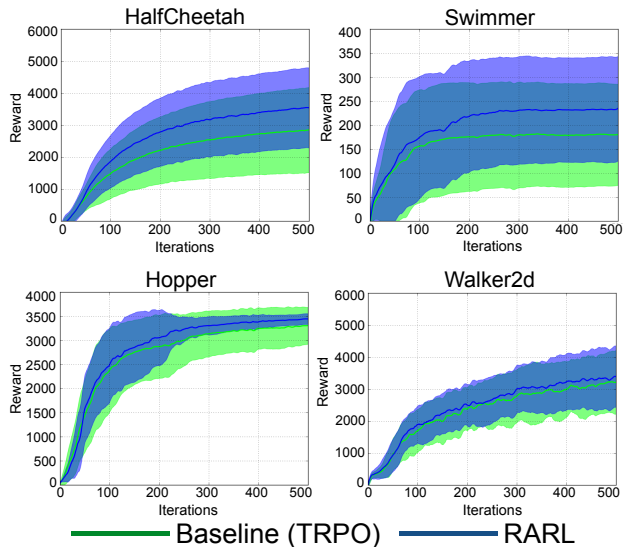

Figure 2. Cumulative reward curves for RARL trained policies versus the baseline (TRPO) when tested without any disturbance. For all the tasks, RARL achieves a better mean than the baseline. For tasks like Hopper, we also see a significant reduction of variance across runs.

However, the primary focus of this paper is to show robustness in training these control policies. One way of visualizing this is by plotting the average rewards for the nth percentile of trained policies. Figure 3 plots these percentile curves and highlight the significant gains in robustness for training for the HalfCheetah, Swimmer and Hopper tasks.

4.3. Robustness under Adversarial Disturbances

While deploying controllers in the real world, unmodeled environmental effects can cause controllers to fail. One way of measuring robustness to such effects is by measuring the performance of our learned control polices in the presence of an adversarial disturbance. For this purpose, we train an adversary to apply a disturbance while holding the protagonist's policy constant. We again show the percentile graphs as described in the section above. RARL's control policy, since it was trained on similar adversaries, performs better, as seen in Figure 4.

Table 1. Comparison of the best policy learned by RARL and the baseline (mean±one standard deviation)

	InvertedPendulum	HalfCheetah	Swimmer	Hopper	Walker2d	Ant
Baseline	1000 ± 0.0	5093 ± 44	358 ± 2.4	3614 ± 2.16	5418 ± 87	5299 ± 91
RARL	1000 ± 0.0	5444 ± 97	354 ± 1.5	3590 ± 7.4	5854 ± 159	5482 ± 28
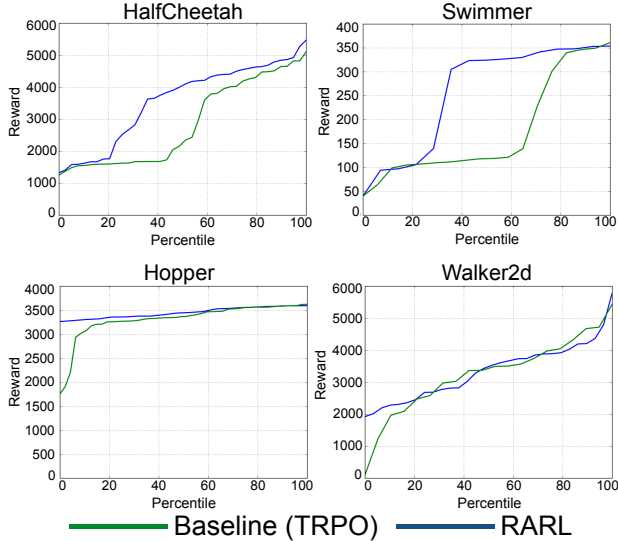
Figure 3. We show percentile plots **without any disturbance** to show the robustness of RARL compared to the baseline. Here the algorithms are run on multiple initializations and then sorted to show the nth percentile of cumulative final reward.
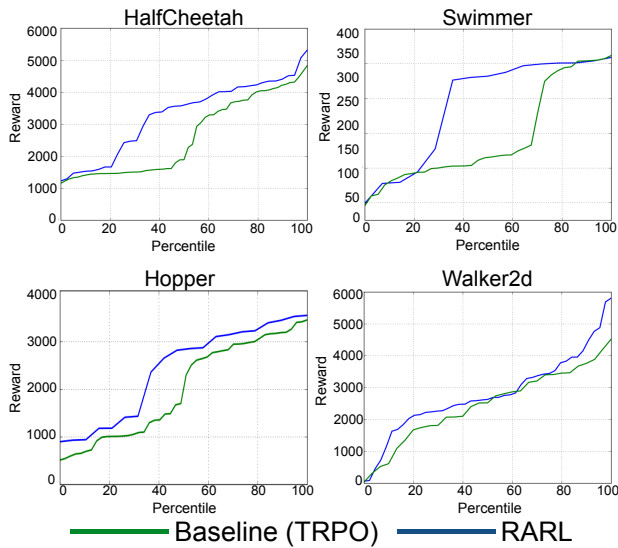
Figure 4. Percentile plots with a **learned adversarial disturbance** show the robustness of RARL compared to the baseline in the presence of an adversary. Here the algorithms are run on multiple initializations followed by learning an adversarial disturbance that is applied at test time.

4.4. Robustness to Test Conditions

Finally, we evaluate the robustness and generalization of the learned policy with respect to varying test conditions. In this section, we train the policy based on certain mass and friction values; however at test time we evaluate the policy when different mass and friction values are used in the environment. Note we omit evaluation of Swimmer since the policy for the swimming task is not significantly impacted by a change mass or friction.

4.4.1. EVALUATION WITH CHANGING MASS

We describe the results of training with the standard mass variables in OpenAI Gym while testing it with different mass. Specifically, the mass of InvertedPendulum, HalfCheetah, Hopper and Walker2D were 4.89, 6.36, 3.53 and 3.53 respectively. At test time, we evaluated the learned policies by changing mass values and estimating the average cumulative rewards. Figure 5 plots the average rewards and their standard deviations against a given torso mass (horizontal axis). As seen in these graphs, RARL policies generalize significantly better.
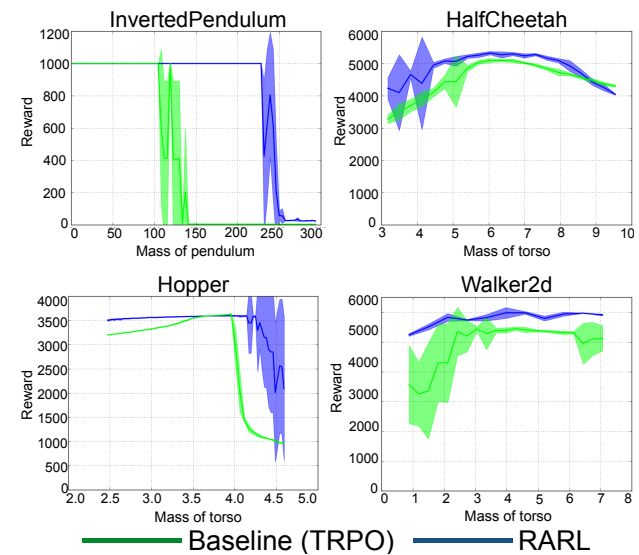

Figure 5. The graphs show robustness of RARL policies to changing mass between training and testing. For the Inverted-Pendulum the mass of the pendulum is varied, while for the other tasks, the mass of the torso is varied. We exclude the results of Ant since it's policies aren't significantly affected by changing mass.

4.4.2. EVALUATION WITH CHANGING FRICTION

Since several of the control tasks involve contacts and friction (which is often poorly modeled), we evaluate robustness to different friction coefficients in testing. Similar to the evaluation of robustness to mass, the model is trained with the standard variables in OpenAI Gym. Figure 6 shows the average reward values with different friction coefficients at test time. It can be seen that the baseline policies fail to generalize and the performance falls significantly when the test friction is different from training. On the other hand RARL shows more resilience to changing friction values.
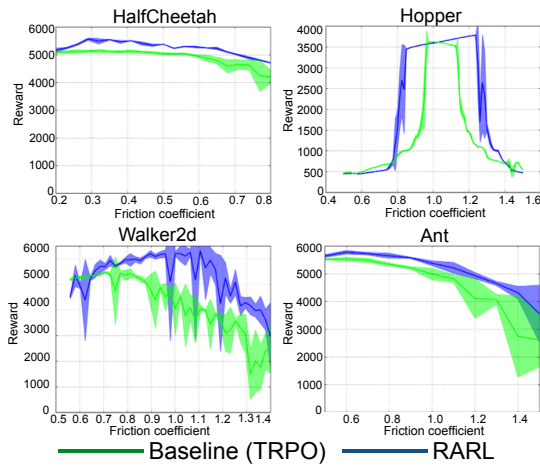

Figure 6. The graphs show robustness of RARL policies to changing friction between training and testing. Note that we exclude the results of InvertedPendulum and the Swimmer because friction is not relevant to those tasks.

We visualize the increased robustness of RARL in Figure 7, where we test with jointly varying both mass and friction coefficient. As observed from the figure, for most combinations of mass and friction values RARL leads significantly higher reward values compared to the baseline.

4.5. Visualizing the Adversarial Policy

Finally, we visualize the adversarial policy for the case of InvertedPendulum and Hopper to see whether the learned policies are human interpretable. As shown in Figure 8, the direction of the force applied by the adversary agrees with human intuition: specifically, when the cart is stationary and the pole is already tilted (top row), the adversary attempts to accentuate the tilt. Similarly, when the cart is moving swiftly and the pole is vertical (bottom row), the adversary applies a force in the direction of the cart's motion. The pole will fall unless the cart speeds up further (which can also cause the cart to go out of bounds). Note that the naive policy of pushing in the opposite direction would be less effective since the protagonist could slow the
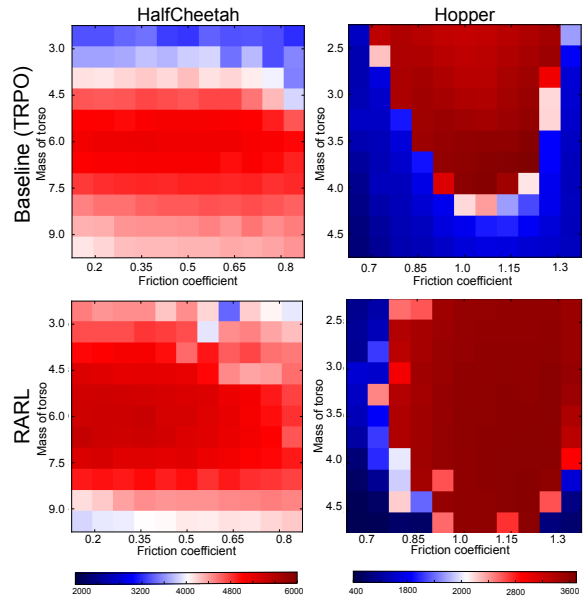

Figure 7. The heatmaps show robustness of RARL policies to changing both friction and mass between training and testing. For both the tasks of Hopper and HalfCheetah, we observe a significant increase in robustness.
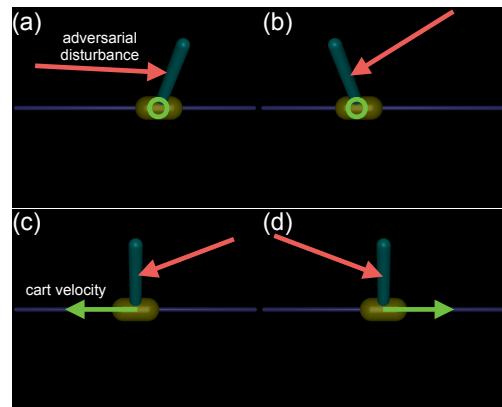
cart to stabilize the pole.

Figure 8. Visualization of forces applied by the adversary on InvertedPendulum. In (a) and (b) the cart is stationary, while in (c) and (d) the cart is moving with a vertical pendulum.

Similarly for the Hopper task in Figure 9, the adversary applies horizontal forces to impede the motion when the Hopper is in the air (left) while applying forces to counteract gravity and reduce friction when the Hopper is interacting with the ground (right).

5. Related Research

Recent applications of deep reinforcement learning (deep RL) have shown great success in a variety of tasks rang-
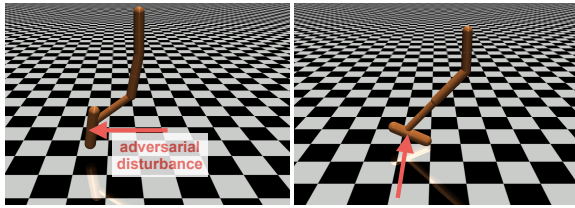
Figure 9. Visualization of forces applied by the adversary on Hopper. On the left, the Hopper's foot is in the air while on the right the foot is interacting with the ground.

ing from games (Mnih et al., 2015; Silver et al., 2016), robot control (Gu et al., 2016; Lillicrap et al., 2015; Mordatch et al., 2015), to meta learning (Zoph & Le, 2016). An overview of recent advances in deep RL is presented in Li (2017) and Kaelbling et al. (1996); Kober & Peters (2012) provide a comprehensive history of RL research.

Learned policies should be robust to uncertainty and parameter variation to ensure predicable behavior, which is essential for many practical applications of RL including robotics. Furthermore, the process of learning policies should employ safe and effective exploration with improved sample efficiency to reduce risk of costly failure. These issues have long been recognized and investigated in reinforcement learning (García & Fernández, 2015) and have an even longer history in control theory research (Zhou & Doyle, 1998). These issues are exacerbated in deep RL by using neural networks, which while more expressible and flexible, often require significantly more data to train and produce potentially unstable policies.

In terms of taxonomy (García & Fernández, 2015), our approach lies in the class of worst-case formulations. We model the problem as an H_∞ optimal control problem (Başar & Bernhard, 2008). In this formulation, nature (which may represent input, transition or model uncertainty) is treated as an adversary in a continuous dynamic zero-sum game. We attempt to find the minimax solution to the reward optimization problem. This formulation was introduced as robust RL (RRL) in Morimoto & Doya (2005). RRL proposes a model-free actor-disturber-critic method. Solving for the optimal strategy for general nonlinear systems is often analytically infeasible for most problems. To address this, we extend RRL's model-free formulation using deep RL via TRPO (Schulman et al., 2015) with neural networks as the function approximator.

Other worst-case formulations have been introduced. Nilim & El Ghaoui (2005) solve finite horizon tabular MDPs using a minimax form of dynamic programming. Using a similar game theoretic formulation Littman (1994) introduces the notion of a Markov Game to solve tabular problems, which involves linear program (LP) to solve

the game optimization problem. Sharma & Gopal (2007) extend the Markov game formulation using a trained neural network for the policy and approximating the game to continue using LP to solve the game. Wiesemann et al. (2013) present an enhancement to standard MDP that provides probabilistic guarantees to unknown model parameters. Other approaches are risk-based including Tamar et al. (2014) and Delage & Mannor (2010), which formulate various mechanisms of percentile risk into the formulation. Our approach focuses on continuous space problems and is a model-free approach that requires explicit parametric formulation of model uncertainty.

Adversarial methods have been used in other learning problems including Goodfellow et al. (2015), which leverages adversarial examples to train a more robust classifiers and Goodfellow et al. (2014); Dumoulin et al. (2016), which uses an adversarial loss function for a discriminator to train a generative model. In Pinto et al. (2017) two supervised agents were trained with one acting as an adversary for self-supervised learning which showed improved robot grasping. Other adversarial multiplayer approaches have been proposed including Heinrich & Silver (2016) to perform self-play or fictitious play. Refer to Buşoniu et al. (2010) for an review of multiagent RL techniques.

Recent deep RL approaches to the problem focus on explicit parametric model uncertainty. Heess et al. (2015) use recurrent neural networks to perform direct adaptive control. Indirect adaptive control was applied in Yu et al. (2017) for online parameter identification. Rajeswaran et al. (2016) learn a robust policy by sampling the worst case trajectories from a class of parametrized models, to learn a robust policy.

6. Conclusion

We have presented a novel adversarial reinforcement learning framework, RARL, that is: (a) robust to training initializations; (b) generalizes better and is robust to environmental changes between training and test conditions; (c) robust to disturbances in the test environment that are hard to model during training. Our core idea is that modeling errors should be viewed as extra forces/disturbances in the system. Inspired by this insight, we propose modeling uncertainties via an adversary that applies disturbances to the system. Instead of using a fixed policy, the adversary is reinforced and learns an optimal policy to optimally thwart the protagonist. Our work shows that the adversary effectively samples hard examples (trajectories with worst rewards) leading to a more robust control strategy.

References

Başar, Tamer and Bernhard, Pierre. *H-infinity optimal control and related minimax design problems: a dynamic game approach*. Springer Science & Business Media, 2008.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

Buşoniu, Lucian, Babuška, Robert, and De Schutter, Bart. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pp. 183–221. Springer, 2010.

Christiano, Paul, Shah, Zain, Mordatch, Igor, Schneider, Jonas, Blackwell, Trevor, Tobin, Joshua, Abbeel, Pieter, and Zaremba, Wojciech. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

Delage, Erick and Mannor, Shie. Percentile optimization for Markov decision processes with parameter uncertainty. *Operations research*, 58(1):203–213, 2010.

Duan, Yan, Chen, Xi, Houthooft, Rein, Schulman, John, and Abbeel, Pieter. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

Dumoulin, Vincent, Belghazi, Ishmael, Poole, Ben, Lamb, Alex, Arjovsky, Martin, Mastropietro, Olivier, and Courville, Aaron. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

Erez, Tom, Tassa, Yuval, and Todorov, Emanuel. Infinite horizon model predictive control for nonlinear periodic tasks. *Manuscript under review*, 4, 2011.

Garcıa, Javier and Fernández, Fernando. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Neural Information Processing Systems (NIPS)*, 2014.

Goodfellow, Ian J, Shlens, Jonathon, and Szegedy, Christian. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR)*, 2015.

Gu, Shixiang, Lillicrap, Timothy, Sutskever, Ilya, and Levine, Sergey. Continuous deep Q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*, 2016.

Heess, Nicolas, Hunt, Jonathan J, Lillicrap, Timothy P, and Silver, David. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

Heinrich, Johannes and Silver, David. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Kaelbling, Leslie Pack, Littman, Michael L, and Moore, Andrew W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Kakade, Sham. A natural policy gradient. *Advances in neural information processing systems*, 2:1531–1538, 2002.

Kober, Jens and Peters, Jan. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pp. 579–610. Springer, 2012.

Li, Yuxi. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Littman, Michael L. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pp. 157–163, 1994.

Mnih, Volodymyr et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mordatch, Igor, Lowrey, Kendall, Andrew, Galen, Popovic, Zoran, and Todorov, Emanuel V. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3132–3140, 2015.

Morimoto, Jun and Doya, Kenji. Robust reinforcement learning. *Neural computation*, 17(2):335–359, 2005.

Nilim, Arnab and El Ghaoui, Laurent. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

Patek, Stephen David. *Stochastic and shortest path games: theory and algorithms*. PhD thesis, Massachusetts Institute of Technology, 1997.

Perolat, Julien, Scherrer, Bruno, Piot, Bilal, and Pietquin, Olivier. Approximate dynamic programming for two-player zero-sum games. In *ICML*, 2015.

Pinto, Lerrel, Davidson, James, and Gupta, Abhinav. Supervision via competition: Robot adversaries for learning tasks. *ICRA*, 2017.

Rajeswaran, Aravind, Ghotra, Sarvjeet, Ravindran, Balaraman, and Levine, Sergey. EPOpt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Rusu, Andrei A, Vecerik, Matej, Rothörl, Thomas, Heess, Nicolas, Pascanu, Razvan, and Hadsell, Raia. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I, and Abbeel, Pieter. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.

Sharma, Rajneesh and Gopal, Madan. A robust Markov game controller for nonlinear systems. *Applied Soft Computing*, 7(3):818–827, 2007.

Shrivastava, Abhinav, Gupta, Abhinav, and Girshick, Ross B. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016.

Silver, David et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Sung, K. and Poggio, T. Learning and example selection for object and pattern detection. *MIT A.I. Memo*, 1521, 1994.

Tamar, Aviv, Glassner, Yonatan, and Mannor, Shie. Optimizing the CVaR via sampling. *arXiv preprint arXiv:1404.3862*, 2014.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Wiesemann, Wolfram, Kuhn, Daniel, and Rustem, Berç. Robust Markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yu, Wenhao, Liu, C. Karen, and Turk, Greg. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

Zhou, Kemin and Doyle, John Comstock. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.