# Reinforcement Learning for Robotics

Erwin M. Bakker
LIACS Media Lab

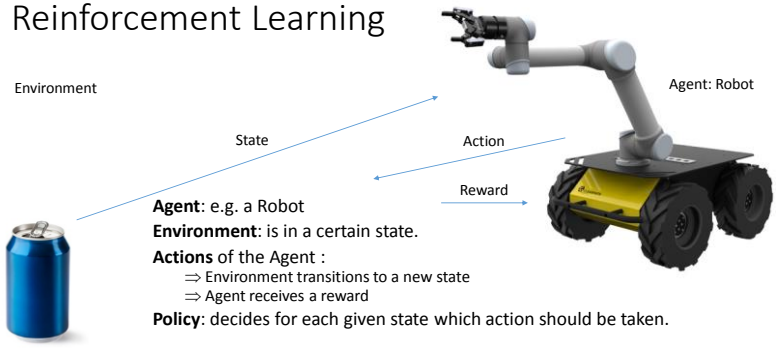
## Reinforcement Learning

E. Charniak, Introduction to Deep Learning. The MIT Press, 2018.

R. Atienza, Advanced Deep Learning with Keras: Apply deep learning techniques, autoencoders, GANs, variational autoencoders, deep reinforcement learning, policy gradients, and more, 2018.

R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series) 2nd Edition, 2018

# Reinforcement Learning




**Agent**: e.g. a Robot

**Environment**: is in a certain state.

**Actions** of the Agent :

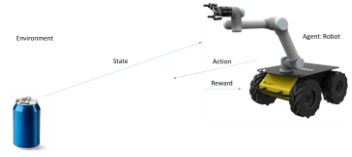
⇒ Environment transitions to a new state

⇒ Agent receives a reward

**Policy**: decides for each given state which action should be taken.

**Goal**: Learn a policy that maximizes the accumulated future rewards

# Markov Decision Process




**Environment**

At time step t the environment is in state $s_t \in S$,
where S is the state space, $s_0$ is the start state, $s_t$ is the end state.

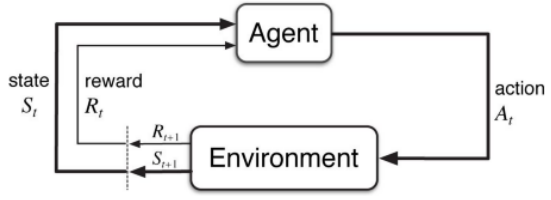
**Actions**

The agent takes actions from the action space A, following a policy $\pi(a_t|s_t)$,
i.e., when the agent finds the environment in state $s_t$, it follows policy $\pi$ and takes action $a_t$ (can be deterministic, or probabilistic)

The environment uses the state transition $T(s_{t+1}|s_t, a_t)$.

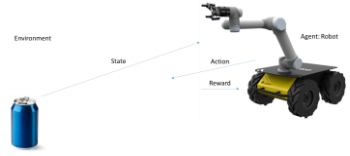
The agent receives a reward $R_{t+1} = R(s_t, a_t)$

## Agent-Environment Interaction




The MDP and Agent give rise to a **trajectory**: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, \ldots$

## Markov Decision Process (MDP)




Environment at time t in state $s_t \in S$.

Action: - $a_t$ following $\pi(a_t|s_t)$

Result: - Environment state transition $T(s_{t+1}|s_t, a_t)$.

- Agent's reward $R_{t+1} = R(s_t, a_t)$

Note:

- T, and R may or may not be known to the agent.
- Future rewards can be discounted by $\gamma^k$, where $\gamma \in [0,1]$, and k a future time step.
- Process can be episodic: then H a horizon is used, with T the number of time steps to complete one episode from $s_0$ to $s_t$.

## Reinforcement Learning (RL)

**Environment**

- Can be fully or partially observable (POMDP)
- For the decision process sometimes past observations are also taken into account.
  - For obeying the **Markov-property**, all information should be maintained in the current state.

**Our robot agent:**

- **State** can be a camera estimate of the 3D position of the soda can with respect to the gripper.
- **Reward**
  - +1, if the robot gets closer to the soda can.
  - -1, if the robot gets farther away from the soda can.
  - +100 when it successfully picks up the soda can.

## Markov Decision Process (MDP) Framework

**Time**

- can be abstract, stages

**Actions**

- can be low-level (voltages applied to a motor in a robot arm) or
- high level (grab lunch, grab can, recharge, etc.)
- Abstract internal

**Environment and States**

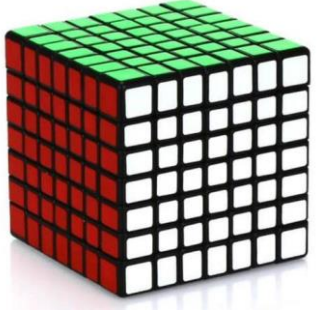
- can be low-level (sensor readings) or
- high level (symbolic descriptions of objects),
- past sensations, subjective, etc.

## Markov Decision Process (MDP) Framework

**Boundary between Environment and Agent:**

- motors, links, and sensors part of environment
- Represents the limit of the agent's absolute control, not of it's knowledge

**Note:** An Agent may know everything about how it's environment works, but still it would be a challenging reinforcement learning task.



## Example: Pick and Place Robot

**Task:**

- Reinforcement Learning to control the motion of a robot arm in a repetitive pick and place task.

**Goal:** fast and smooth movements

**Agent needs:**

- Direct low level control of motors
- Low-latency information of position and velocities of mechanical links

**Actions**

- Voltage applied to each motor at each oint
- Readings of joint angles and velocities

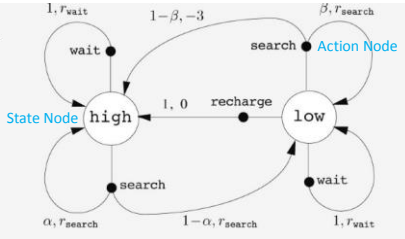
**Reward**

- +1 for each object that is picked and placed
- Small negative reward as function of the jerkiness of the motion (per moment).

## Example: Recycling Robot

transition | action prob. | reward

| $s$ | $a$ | $s'$ | $p(s'\|s,a)$ | $r(s,a,s')$ |
|---|---|---|---|---|
| high | search | high | $\alpha$ | $r_{\text{search}}$ |
| high | search | low | $1-\alpha$ | $r_{\text{search}}$ |
| low | search | high | $1-\beta$ | $-3$ |
| low | search | low | $\beta$ | $r_{\text{search}}$ |
| high | wait | high | 1 | $r_{\text{wait}}$ |
| high | wait | low | 0 | $r_{\text{wait}}$ |
| low | wait | high | 0 | $r_{\text{wait}}$ |
| low | wait | low | 1 | $r_{\text{wait}}$ |
| low | recharge | high | 1 | 0 |
| low | recharge | low | 0 | 0 |




**High level agent decides to search, wait or recharge:**

- Two charge levels: high, low
- Action set: π(high)={search, wait}, π(low) = {search, wait, recharge}

## Goals and Rewards

- Agent receives at each time step a reward $R_t$
- Goal is to maximize the total amount of received rewards.

**The maximization of the expected value of the cumulative sum of a received scalar signal (called reward).**

More formally:
Sequence of rewards after time step t: $R_{t+1}$, $R_{t+2}$, $R_{t+3}$, …
T final time step, sum of rewards $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$

## Reinforcement Learning (RL)

**Goal:**

- Maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{T+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad \gamma \in [0,1]$$

Note:

- $\gamma \in [0,1]$ the discount rate.
- $\gamma = 0$, if the immediate reward matters
- $\gamma = 1$, if future rewards weigh the same as the immediate reward

## Reinforcement Learning (RL)

Goal:

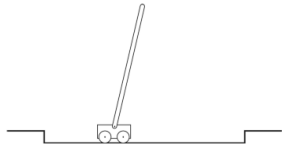
- Maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{T+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad \gamma \in [0,1]$$

Note:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{T+3} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{T+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

## Example: Pole-Balancing



**Objective:** Apply forces to the cart such that pole does not fall over.
Failure: If pole falls, or cart runs off the track.

**Task of pole-balancing seen as repeated attempts, episodes, during which it is balanced:**
Reward: +1 for every time step without failure
⇒reward = ∞ if successful balancing for ever.

**Pole-balancing seen as a continuous task:**
Reward: -1 on each failure, 0 otherwise.
=> reward related to $-\gamma^K$, where K is the number of time steps before failure.

## Policies and Value Functions

**Try to estimate value-functions (of states, or state-action pairs) that estimate for an agent:**

- how good it is to be in a state or
- how good it is to perform a given action in given state

The value function of a state s under a policy $\pi$ is defined as:
$v_\pi(s) = \boldsymbol{E_\pi}[\boldsymbol{G_t}|\boldsymbol{S_t} = \boldsymbol{s}] = \mathrm{E}_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s]$, for all $s \in S$

The expected return starting from s, taking action a and further on following policy $\pi$ is defined as:

$$q_\pi(s,a) = \boldsymbol{E_\pi}[\boldsymbol{G_t}|\boldsymbol{S_t} = \boldsymbol{s}, \boldsymbol{A_t} = \boldsymbol{a}] = \mathrm{E}_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

## Reinforcement Learning (RL)

**Goal:**

- Learn an optimal policy $\pi^*$, where

$$\pi^* = argmax_\pi G_t, \quad \text{where } G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}, \quad \gamma \in [0,1],$$

and $R_{t+1} = R(s_t, a_t)$

**Methods:**

- Brute Force, Tabular Methods, Monte Carlo Methods, DNN for RL, Adversarial RL.

[1] L. Pinto, J. Davidson, R. Sukthankar, A. Gupta, Robust Adversarial Reinforcement Learning, March 2017.

Deep neural networks success in the field of Reinforcement Learning:

- Fast computations
- Fast Simulations
- Improved networks

But, most RL-based approaches fail to generalize, because:

1. gap between simulation and real world
2. policy learning in real world is hampered by data scarcity

## RL Challenges for Real-world Policy Learning


The training of the agent's policy in the real-world:

- too expensive
- dangerous
- time-intensive

=> scarcity of data.

=> training often restricted to a limited set of scenarios, causing overfitting.

=> If the test scenario is different (e.g., different friction coefficient, different mass), the learned policy fails to generalize.

But a learned policy should be robust and generalize well for different scenarios.

## RL in the Real World: use more robots

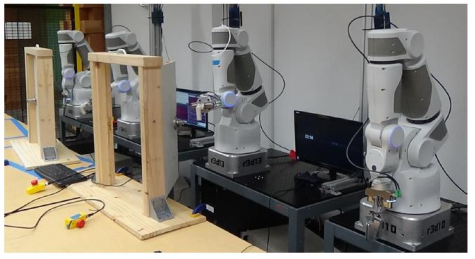


Fig. 1: Two robots learning a door opening task. We present a method that allows multiple robots to cooperatively learn a single policy with deep reinforcement learning.

From [2] Gu et al. , Nov. 2016.

## Reinforcement Learning in simulation:

Facing the **data scarcity** in the real-world by
- Learning a policy in a simulator
- Transfer learned policy to the real world

But the environment and physics of the simulator
are not the same as the real world.

**=> Reality Gap**

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).

## Robust Adversarial Reinforcement Learning (RARL)

Training of an agent in the presence of a **destabilizing adversary**

- Adversary can employ disturbances to the system
- Adversary is trained at the same time as the agent
- Adversary is reinforced: it learns an optimal destabilization policy.

Here policy learning can be formulated as a zero-sum, minimax objective function.

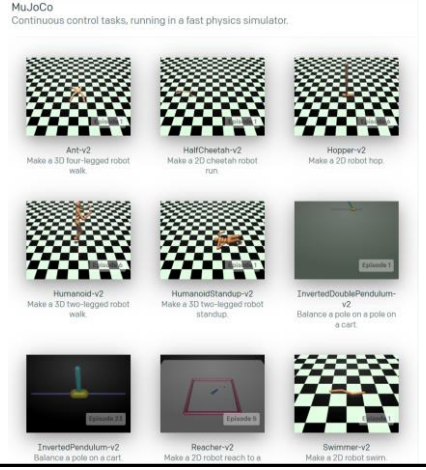
Minimax in zero-sum games denotes minimizing the opponent's maximum payoff.
In a zero-sum game, this is identical to:
- minimizing one's own maximum loss, and to
- maximizing one's own minimum gain

Zero-sum game: gain and loss cancel each other out.

## Experimental Environments

- InvertedPendulum
- HalfCheetah
- Swimmer
- Hopper
- Walker2d

https://gym.openai.com/




## Unconstrained Scenarios: Challenges

In unconstrained scenarios:

- the space of possible disturbances could be larger than the space of possible actions

=> sampled trajectories for learning etc. even sparser

## Challenges of unconstrained scenarios

**Use adversaries for modeling disturbances:**

- we do not want to and can not sample all possible disturbances
- we jointly train a second agent (**the adversary**)
- goal of adversary is to impede the original agent (**the protagonist**)
  - by applying destabilizing forces.
  - rewarded only for the failure of the protagonist

  => the adversary learns to sample hard examples, disturbances that make original agent fail
  - the protagonist learns a policy that is robust to any disturbances created by the adversary.

## Challenges of unconstrained scenario


**Use adversaries that incorporate domain knowledge:**

- Naïve: **give adversary the same action space as the protagonist**
  - Like a driving student and driving instructor fighting for control of a dual-control car.

**Proposal paper:**

- exploit **domain knowledge**
- focus on the protagonist's weak points;
- give the adversary "super-powers" – the ability to affect the robot or environment in ways the protagonist cannot (e.g. suddenly change frictional coefficient or mass).

## Adversary with Domain Knowledge

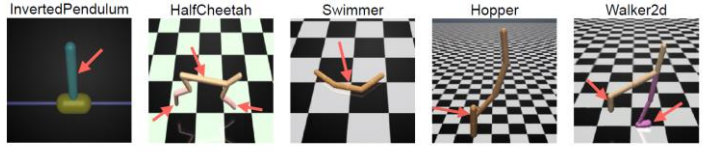



*Figure 1.* We evaluate RARL on a variety of OpenAI gym problems. The adversary learns to apply destabilizing forces on specific points (denoted by red arrows) on the system, encouraging the protagonist to learn a robust control policy. These policies also transfer better to new test environments, with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

Figure from [1].

## Standard Reinforcement Learning (RL)

RL for continuous space Markov Decision Processes

$(S, A, P, r, \gamma, s_0)$, where
$S$ the set of continuous states
$A$ the set of continuous actions
$P: S \times A \times S \rightarrow \mathbb{R}$ the transition probability
$r: A \rightarrow \mathbb{R}$ the reward function
$\gamma$ the discount factor
$s_0$ the initial state distribution

## Standard Reinforcement Learning (RL)

- RL for continuous space Markov Decision Processes
  - $(S, A, P, r, \gamma, s_0)$, where
  - $S$ the set of continuous states
  - $A$ the set of continuous actions
  - $P: S \times A \times S \rightarrow \mathbb{R}$ the transition probability
  - $r: S \times A \rightarrow \mathbb{R}$ the reward function
  - $\gamma$ the discount factor
  - $s_0$ the initial state distribution

Batch policy algorithms [Williams 1992, Kakade 2002, Shulman 2015]:

Learning a stochastic policy:

$\pi_\theta: S \times A \rightarrow \mathbb{R}$ which maximizes $\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$

the cumulative discounted reward

- $\Theta$ the parameters of the policy $\pi$.
- Policy $\pi$ takes action $a_t$ given state $s_t$ at time $t$

## 2 Player $\gamma$ discounted zero-sum Markov Game

(Litman 1994, Perolat 2015)

- 2 Player continuous space Markov Decision Processes
  $(S, A_1, A_2, P, r, \gamma, s_0)$, where
  $S$ the set of continuous states
  $A$ the set of continuous actions
  $P: S \times A_1 \times A_2 \times S \rightarrow \mathbb{R}$ the transition probability
  $r: S \times A_1 \times A_2 \rightarrow \mathbb{R}$ the reward function of both players
  $\gamma$ the discount factor
  $s_0$ the initial state distribution

Let Player 1 use strategy $\mu$ and Player 2 use strategy $\vartheta$, then the reward function is given by:

$$r_{\mu,\vartheta} = E_{a^1 \sim \mu(.|S), a^2 \sim \vartheta(.|S)}[r(s, a^1, a^2)]$$

**Player 1 will be maximizing while Player 2 minimizes the $\gamma$ discounted reward.**

(=> Zero Sum 2 player game)

## RALR Algorithm

The initial parameters for both players' policies are sampled from a random distribution.

**Two phases**

1. Learn the protagonist's policy while holding the adversary's policy fixed.
2. The protagonist's policy is held constant and the adversary's policy is learned.

Repeat until convergence.

In each phase a *roll*-function is used sampling the $N_{traj}$ trajectories in environment $\mathcal{E}$.

$\mathcal{E}$ contains the transition function $P$ and reward functions $r^1$ and $r^2$

**Algorithm 1** RARL (proposed algorithm)

**Input:** Environment $\mathcal{E}$; Stochastic policies $\mu$ and $\nu$
**Initialize:** Learnable parameters $\theta_0^\mu$ for $\mu$ and $\theta_0^\nu$ for $\nu$
**for** $i$=1,2,...$N_{\text{iter}}$ **do**
  $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$
  **for** $j$=1,2,..$N_\mu$ **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,..$N_\nu$ **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:** $\theta_{N_{\text{iter}}}^\mu, \theta_{N_{\text{iter}}}^\nu$

# Experimental Setup

Environments built using OpenAI gym's (Brockman et al., 2016).
Control of environments with the MuJoCo physics simulator (Todorov et al., 2012) .

RARL is built on top of rllab (Duan et al., 2016)
Baseline: Trust Region Policy Optimization (TRPO) (Schulman et al., 2015)

For all the tasks and for both the protagonist and adversary,
a policy network with two hidden layers with 64 neurons per layer is used.

RARL and the baseline are trained with
- 100 iterations on InvertedPendulum
- 500 iterations on the other environments

Hyperparameters of TRPO are selected by grid search.


**InvertedPendulum**
- State space 4D: position, velocity
- Protagonist: 1D forces; Adversary: 2D forces on center of pendulum

**HalfCheetah**
- State space 17D: joint angles and joint velocities, …
- Adversary: 6D actions with 2D forces

**Swimmer**
- State space 8D: joint angles and joint velocities, …
- Adversary: 3D forces to center of swimmer

**Hopper**
- State space 11D: joint angles and joint velocities, …
- Adversary: 2D force on foot

**Walker2d**
- State space 17D: joint angles and joint velocities, …
- Adversary: 4D actions with 2D forces on both feet

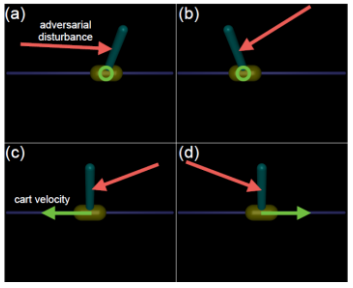



*Figure 8.* Visualization of forces applied by the adversary on InvertedPendulum. In (a) and (b) the cart is stationary, while in (c) and (d) the cart is moving with a vertical pendulum.

## Actions of Adversary

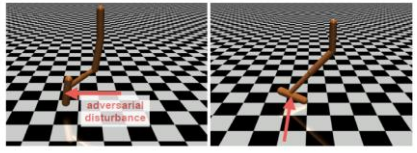
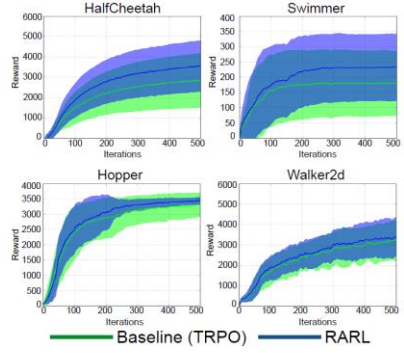



*Figure 9.* Visualization of forces applied by the adversary on Hopper. On the left, the Hopper's foot is in the air while on the right the foot is interacting with the ground.




## Results

*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.

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

| | InvertedPendulum | HalfCheetah | Swimmer | Hopper | Walker2d |
|---|---|---|---|---|---|
| Baseline | $1000 \pm 0.0$ | $5093 \pm 44$ | $358 \pm 2.4$ | $3614 \pm 2.16$ | $5418 \pm 87$ |
| RARL | $1000 \pm 0.0$ | $5444 \pm 97$ | $354 \pm 1.5$ | $3590 \pm 7.4$ | $5854 \pm 159$ |

Results
Robustness to
Changing Mass

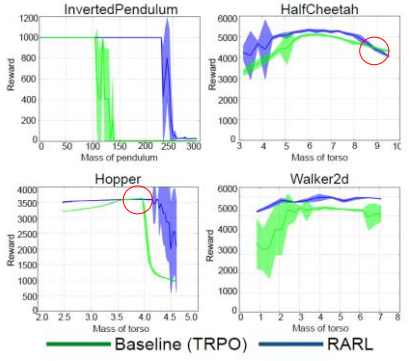



*Figure 5.* The graphs show robustness of RARL policies to changing mass between training and testing. For the Inverted-Pendulum the mass of the pendulum is varied, while for the other tasks, the mass of the torso is varied.

Results Robustness to Changing Friction

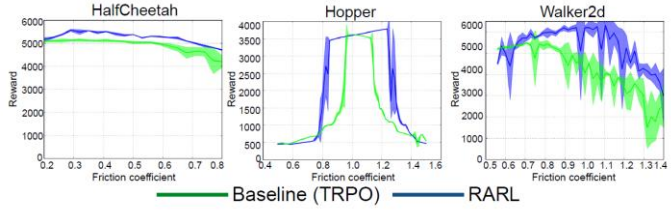


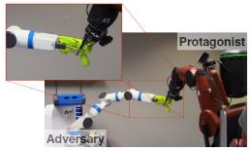

*Figure 6.* The graphs show robustness of RARL policies to changing friction between training and testing. Note that we exclude the results of InvertedPendulum and the Swimmer because friction is not relevant to those tasks.

## Conclusions Experiment Results

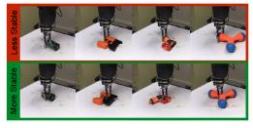
1. improves training stability
2. is robust to differences in training/test conditions
3. outperform the baseline even in the absence of the adversary

## Discussion

- Results for completely simulated environments: how does it translate to the real world?
- Adversary can be very easily too powerful. How do you incorporate/ formulate the adversary's powers in your RARL model?
- Can you think of a good hybrid setup: part simulator, part the real thing. Have the adversary coming from/to the real world into the simulation…
- …




From [4] Pinto et al., 2016.



# References


1. L. Pinto, J. Davidson, R. Sukthankar, A. Gupta, Robust Adversarial Reinforcement Learning, arXiv:1703.02702, March 2017.
2. S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates, arXiv:1610.00633v2 [cs.RO], October 2016.
3. C. Finn, S. Levine, Deep Visual Forsight for Planning Robot Motion, arXiv:1610.00696, ICRA 2017, October 2016.
4. L. Pinto, J. Davidson, A. Gupta, Supervision via Competition: Robot Adversaries for Learning Tasks, arXiv:1610.01685, ICRA 2017, October 2016.
5. K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, D. Krishnan, Unsupervised Pixel–Level Domain Adaptation with Generative Adversarial Networks, arXiv:1612.05424, CVPR 2017, December 2016.
6. A. Banino et al., Vector-based navigation using grid-like representations in artificial agents, https://doi.org/10.1038/s41586-018-0102-6, Research Letter, Nature, 2018.
7. R. Borst, Robust self-balancing robot mimicking, Bachelor Thesis, August 2017