Active Domain Randomization

Anonymous Authors[1]

Abstract

Domain randomization is a popular technique for zero-shot domain transfer, often used in reinforcement learning when the target domain is unknown or cannot easily be used for training. In this work, we empirically examine the effects of domain randomization on agent generalization and sample complexity. Our experiments show that domain randomization may lead to suboptimal policies even in simple simulated tasks, which we attribute to the uniform sampling of environment parameters. We propose Active Domain Randomization, a novel algorithm that learns a sampling strategy of randomization parameters. Our method looks for the most informative environment variations within the given randomization ranges by leveraging the differences of policy rollouts in randomized and reference environment instances. We find that training more frequently on these proposed instances leads to faster and better agent generalization. In addition, when domain randomization and policy transfer fail, Active Domain Randomization offers more insight into the deficiencies of both the chosen parameter ranges and the learned policy, allowing for more focused debugging. Our experiments across various physics-based simulated tasks show that this enhancement leads to more robust policies, all while improving sample efficiency over previous methods.

1. Introduction

Recent trends in Deep Reinforcement Learning (DRL) exhibit a growing interest for zero-shot domain transfer, i.e. when a policy is learned in a source domain and is then tested *without finetuning* in an unseen target domain. Zero-shot transfer is particularly useful when the task in the target domain is inaccessible, complex, or expensive, such as gathering rollouts from a real-world robot. An ideal agent would learn to *generalize* across domains; it would accomplish the task without exploiting irrelevant features or deficiencies in the source domain (i.e., approximate physics in simulators), which may vary dramatically after transfer. Any agent that fails this transfer task falls prey to the *domain adaptation* problem.

One promising route to zero-shot transfer has been *domain randomization* (Tobin et al., 2017). The approach is simple: when episodically training a policy in a simulator, uniformly randomize every environment parameter of the simulation (e.g. friction, motor torque) across predefined ranges. By randomizing everything that might vary in the target environment, the hope is that eventually, the target domain will be just another variation. Yet, domain randomization is not without its flaws. Recent works suggest that the sample complexity grows exponentially in terms of the number of randomization parameters, even when dealing only with transfer between simulations (e.g., in Andrychowicz et al. (2018) Figure 8). In addition, when using domain randomization *unsuccessfully*, policy transfer fails as a black box. After a failed transfer, randomization ranges are tweaked heuristically via trial-and-error. Repeating this process iteratively, researchers are often left with arbitrary ranges that do (or do not) lead to policy convergence without any insight into how those settings may be beneficial or detrimental to the learned behavior.

In this work we investigate the impact of parameter sampling for domain randomization. We show that, in a reinforcement learning setting, uniform sampling of environment parameters is suboptimal, and moreover, that the generalization performance of the learned policy is much more sensitive to some parameters than others. This motivates the development of our algorithm, Active Domain Randomization (ADR), which has the following benefits (which we claim as our contributions):

1. ADR learns the most informative variations in the randomization space. Briefly, ADR searches for randomization settings where the agent policy deviates most from its behavior in a reference environment. We find that such environment instances correspond to harder versions of the problem and that prioritizing these samples in training leads to faster and better generalization.
2. The learned sampling strategy of ADR is reusable and

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

can be extracted to bootstrap new agents even more efficiently while still maintaining the benefits of generalization.

3. ADR can provide insight into which dimensions and parameter ranges are most influential *before transfer*, which can alert researchers of overfitting or simulation flaws before expensive experiments are undertaken.

We illustrate how ADR is applicable to a broad spectrum of domain adaption problems, by showcasing its benefits on a variety of simulated, continuous control benchmarks.

2. Background

Our method leverages Reinforcement Learning (RL) in two contexts - one to train the agent to perform the task in the environment and one to generate parameter samples for environment instances for domain randomization. In the latter, we use a method based on Stein Variational Policy Gradient (SVPG), which is useful for efficiently exploring high dimensional state spaces.

2.1. Reinforcement Learning

We consider a RL framework (Sutton & Barto, 2018) where some task T is defined by a Markov Decision Process (MDP) consisting of a state space S, action space A, state transition function $P : S \times A \mapsto S$, and reward function $R : S \times A \mapsto \mathbb{R}$. The goal for an agent trying to solve T is to learn a policy π with parameters θ that maximizes the expected total discounted reward

$$J(\pi) = \mathbb{E}_{(s_t,a_t)\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] \tag{1}$$

where $\gamma \in (0, 1)$ is the discount factor. We define a rollout $\tau = (s_0, a_0..., s_T)$ to be the states s_t seen by policy π when executing actions $a_t \sim \pi_\theta(a_t|s_t)$ in the environment. For more details on RL basics please refer to Sutton & Barto (2018).

2.2. Domain Randomization

Domain Randomization (DR) is a technique introduced to overcome the domain adaptation issue, especially when training policies completely in simulation and transferring them in a zero-shot manner to the real world. DR requires a prescribed set of N_{rand} simulation parameters to randomize, as well as corresponding ranges to sample them from. This induces the notion of a *randomization space* $\Xi \subset \mathbb{R}^{N_{rand}}$, where each randomization parameter $\xi^{(i)}$ is bounded on a closed interval $\{[\xi^{(i)}_{low}, \xi^{(i)}_{high}]\}_{i=1}^{N_{rand}}$. When a configuration $\xi \in \Xi$ is passed to a non-differentiable simulator S, it generates an environment E, which the agent policy π_θ sees and uses to train.

Algorithm 1 Uniform Sampling Domain Randomization

Input: Randomization Space Ξ, Simulator S
Initialize agent policy π_θ
for each episode **do**
  // Uniformly sample parameters
  for $i = 1$ **to** N_{rand} **do**
    $\xi^{(i)} \sim U[\xi^{(i)}_{low}, \xi^{(i)}_{high}]$
  end for
  // Generate, rollout in randomized env.
  $E_i \leftarrow S(\xi_i)$
  rollout $\tau_i \sim \pi_\theta(\cdot; E_i)$
  $\mathcal{T}_{rand} \leftarrow \mathcal{T}_{rand} \cup \tau_i$
  for each gradient step **do**
    // Agent policy update
    with $\mathcal{T}_{rand}$ **update**:
      $\theta \leftarrow \theta + \nu \nabla_\theta J(\pi_\theta)$
  end for
end for

Generally, at the start of each episode, the parameters are uniformly sampled from the ranges, and the environment generated from those values is passed to the agent policy π_θ for training. DR changes any to all parts of the task T's underlying MDP[1], with the exception of keeping R constant. DR therefore generates a multitude of MDPs that are superficially similar, but can vary greatly in difficulty depending on the character of the randomization. Upon transfer to the target domain, the expectation is that policy has learned to generalize across MDPs, and sees the final domain as just another variation.

The most common instantiation of DR, Uniform-Sampling Domain Randomization (USDR) is summarized in Algorithm 1. USDR generates randomized environment instances E_i by uniformly sampling the randomization space. The agent policy π_θ is then trained on rollouts τ_i produced in those randomized environments.

2.3. Stein Variational Policy Gradient

Sufficient exploration in high-dimensional state spaces has always been a difficult problem in RL. Recently, Liu et al. (2017) proposed SVPG, aims to learn a policy μ_ϕ in a Maximum-Entropy RL framework (Ziebart, 2010).

$$\max_{\mu} \mathbb{E}_\mu[J(\mu)] + \alpha\mathcal{H}(\mu) \tag{2}$$

with entropy $\mathcal{H}$ being controlled by temperature parameter α. SVPG uses Stein Variational Gradient Descent (Liu &

[1]The effects of DR on action space A are usually implicit or are carried out on the simulation side.

Wang, 2016) to iteratively update an ensemble of N policies or *particles* $\mu_\phi = \{\mu_{\phi_i}\}_{i=1}^N$ with an update rule:

$$\mu_{\phi_i} \leftarrow \mu_{\phi_i} + \epsilon \Delta\mu_{\phi_i}$$
$$\Delta\mu_{\phi_i} = \frac{1}{N}\sum_{j=1}^{N}[\nabla_{\mu_{\phi_j}} J(\mu_{\phi_j})k(\mu_{\phi_j}, \mu_{\phi_i}) + \alpha\nabla_{\mu_{\phi_j}} k(\mu_{\phi_j}, \mu_{\phi_i})] \quad (3)$$

with step size ϵ and positive definite kernel k. This update rule balances exploitation (the first term moves particles towards high-reward regions) and exploration (the second term repulses similar policies). In our work we use an Radial Basis Function (RBF) kernel with adaptive baseline as described in Liu et al. (2017) and an Advantage Actor-Critic (A2C) policy gradient estimator (Mnih et al., 2016), although both the kernel and estimator could be substituted with alternative methods (Gangwani et al., 2019).

3. Method

3.1. Motivating Experiment

To motivate the need for a better sampling strategy, we begin by investigating the validity of the following claim: *uniformly sampling of environment parameters does not generate equally useful MDPs*. We do so by performing an experiment on a toy environment, `LunarLander-v2`, where the agent's task is to ground a lander in a designated zone (Figure 3-left), and is rewarded based on the quality of landing (fuel used, impact velocity, etc). Parameterized by an 8D state vector and actuated by a 2D continuous action space, `LunarLander-v2` has one main axis of randomization that we vary: the main engine strength (MES).

Targeting the uniform sampling strategy in DR, we aim to determine if certain environment instances (different values of the MES) are more *informative* - more efficient than others in terms of aiding learning speed and generalization. We set the total range of variation for the MES to be $[8, 20]$ (the default is 13, and lower than 7.5 makes the environment practically unsolvable when all other physics parameters are held constant) and find through simple tests that lower engine strengths generate harder MDPs to solve. Under this assumption, we show the effects of *focused domain randomization* by editing the ranges that the main engine strength is uniformly sampled from.

We train multiple agents, with the only difference between them being the randomization ranges for MES during training. The randomization ranges define what types of environments the agent sees during training. Figure 1 shows the final generalization performance of each agent by sweeping across the entire randomization range of [8, 20] and rolling out the policy in the generated environments. We see that focusing on harder MDPs improves generalization over traditional DR, even when the evaluation environment is outside of the training distribution.

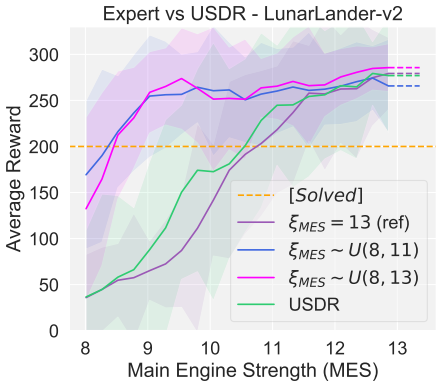


Figure 1. Generalization for various agents who saw different main engine strength (MES) randomization ranges.

3.2. Active Domain Randomization

While interesting, the experiment in the previous section is problematic for two reasons:

1. It is rare that such intuitively *hard* MDP instances or parameter ranges are known beforehand.
2. DR is used mostly when the space of randomized parameters is high dimensional (noninterpretable).

To address these two issues, we extend our initial intuitions: not only are *all parts of a randomization range* not equally informative, but also individual *randomization dimensions* are not equally useful to focus on. In `LunarLander-v2`, the main engine strength is more crucial to task performance than the side engine strength, which only marginally affects generalization performance when varied. An ideal randomization scheme would learn this inherent difference between the dimensions' values and focus on the most difficult environment instances rather than sampling uniformly across the entire space.

To this end we propose ADR, summarized in Algorithm 2 and Figure 3.2. ADR tries to find these informative MDPs within the randomization space by formulating the search as an RL problem. ADR attempts to learn a policy μ_ϕ where the states are proposed randomization configurations $\xi \in \Xi$ and actions are changes to those parameters.

We learn a discriminator-based reward for μ_ϕ, similar to the one originally proposed in Eysenbach et al. (2018):

$$r_D = \log D_\psi(y|\tau_i \sim \pi_\theta(\cdot; E_i)) \quad (4)$$

where y is an boolean variable denoting the discriminator's

Algorithm 2 Active Domain Randomization

Input: Ξ: randomization space, S: simulator, ξ_{ref}: reference parameters
Initialize π_θ: agent policy, μ_ϕ: SVPG particles, D_ψ: discriminator, $E_{ref} \leftarrow S(\xi_{ref})$: reference environment
while not $max\_timesteps$ **do**
  for each sampling step **do**
    rollout $\xi_i \sim \mu_\phi(\cdot)$
  end for
  for each ξ_i **do**
    // Generate, rollout in randomized env.
    $E_i \leftarrow S(\xi_i)$
    rollout $\tau_i \sim \pi_\theta(\cdot; E_i)$, $\tau_{ref} \sim \pi_\theta(\cdot; E_{ref})$
    $\mathcal{T}_{rand} \leftarrow \mathcal{T}_{rand} \cup \tau_i$
    $\mathcal{T}_{ref} \leftarrow \mathcal{T}_{ref} \cup \tau_{ref}$
    for each gradient step **do**
      // Agent policy update
      with $\mathcal{T}_{rand}$ **update**:
        $\theta \leftarrow \theta + \nu \nabla_\theta J(\pi_\theta)$
    end for
  end for
  // Calculate reward for each proposed environment
  for each $\tau_i \in \mathcal{T}_{rand}$ **do**
    Calculate reward with associated ξ_i and E_i using Eq. (4)
  end for
  // Update randomization sampling strategy
  for each particle μ_{ϕ_i} **do**
    Update particles using Eq. (3)
  end for
  // Update discriminator
  for each gradient step **do**
    Update D_ψ with τ_i and τ_{ref} using SGD.
  end for
end while

prediction of which type of environment (a randomized environment E_i or reference environment E_{ref}) the trajectory τ_i was generated from. In our work, we define the E_{ref} to be the environment defined by the *default* parameter configuration ξ_{ref}, which comes along with the original task definition.

Intuitively, we reward the policy μ_ϕ for finding regions of the randomization space that produce environment instances where the *same* agent policy π_θ acts differently than in the reference environment. The agent policy π_θ sees and trains *only* on the randomized environments (as it would in traditional DR), using the environment's task-specific reward for updates. As the agent improves on the proposed, problematic environments, it becomes more difficult to differentiate whether any given state transition was generated from the reference or randomized environment. Thus, ADR can find what parts of the randomization space the agent is currently performing poorly on, and can *actively* update its sampling strategy throughout the training process.

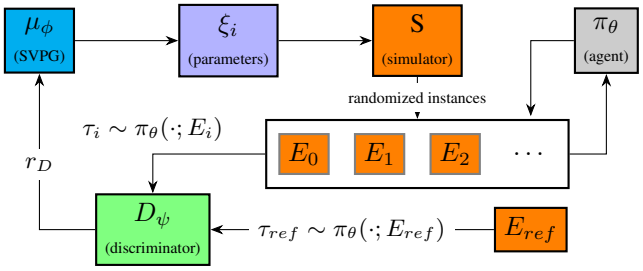


Figure 2. Active Domain Randomization looks for the most informative environment variations by generating randomized instances, and comparing an agent rolled out there to that *same* agent being rolled out in a reference environment.

To encourage sufficient exploration in high-dimensional randomization spaces, we parameterize μ_ϕ with SVPG. SVPG benefits from multiple, diverse policies and gives the agent policy π_θ the same type of environment variety seen in DR while still using the learned reward to hone in on problematic MDP instances. In addition, SVPG is highly parallelizable, which allows us to gather agent rollouts across various randomized environments simultaneously (Lines 7 to 18 of Algorithm 2).

4. Results

4.1. Implementation Details

To test ADR, we experiment on OpenAI Gym environments (Brockman et al., 2016) across various physics engines shown in Figure 3:

- `LunarLander-v2`[2], a 2 degrese of freedom (DoF) environment in which the agent has to softly land a spacecraft, implemented in Box2D (detailed in Section 3.2),
- `Pusher-3DOF-v0`[3], a 3 DoF arm that has to push a puck to a target, implemented in Mujoco (Todorov et al., 2012), and
- `ErgoReacher-v0`[4], a 4 DoF arm which has to touch a goal with its end effector, implemented in the Bullet Physics Engine (Coumans, 2015).

In addition to the randomization of the main engine strength of `LunarLander-v2`, both other environments randomize various physics parameters that change the environments

[2] https://gym.openai.com/envs/LunarLander-v2/
[3] Originally developed for Haarnoja et al. (2018)
[4] Originally developed for Golemo et al. (2018)

drastically. `Pusher-3DOF-v0` has two axes of randomization that make the puck slide more or less when pushed, and `ErgoReacher-v0` randomized the max torque and gain for each degree of freedom (joint), which accounts for eight randomization parameters. We provide a detailed account of the randomizations used in Table **??** within the supplementary material.

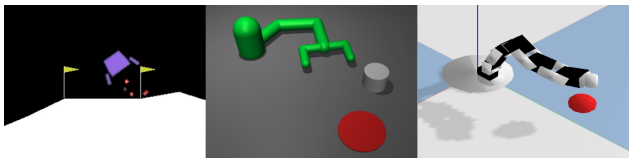


Figure 3. Our testing tasks include `LunarLander-v2` (left), a 3 DoF Pusher environment (middle), and a 4 DoF `ErgoReacher` environment (right)

Across all experiments, our agent policy is trained with Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), although the agent policy can be substituted with any other on or off-policy algorithm with only minor changes to Algorithm 2. Our discriminator-based reward generator is a three layers, 128-neurons, Fully Connected Neural Network (FCN).

The simulator parameter sampling strategy is parameterized by SVPG with RBF Kernel and temperature $\alpha = 10$, and we use A2C to calculate unbiased and low variance gradient estimates. Each of the SVPG particles is also an FCN of two layers of 100 neurons each. All experiments results are plotted (mean) averaged across five seeds, five trials per evaluation point, with one standard deviation shown. Detailed experiment, environment, and architectural information can be found in the supplementary material and attached code.

4.2. Toy Experiments

To investigate whether ADR's learned sampling strategy provides a tangible benefit in both agent generalization and learning speed, we start by comparing it against traditional DR (labeled as USDR) on `LunarLander-v2` and vary only the main engine strength (MES). In Figure 4, we see that ADR approaches expert-levels of generalization whereas traditional DR fails to generalize on lower MES ranges.

From Figure 5(a), we see that not only does ADR solve the reference environment ($\xi_{MES} = 13$) more quickly than DR, but also more consistently throughout training. Figure 5(b) shows that on the *hard* environment instances described in Section 3.1 ($\xi_{MES} \sim U[8, 11]$), ADR is the only agent out of both the baseline (trained only on MES of 13) and the USDR agent (trained seeing environments with $\xi_{MES} \sim U[8, 20]$) to make significant progress.

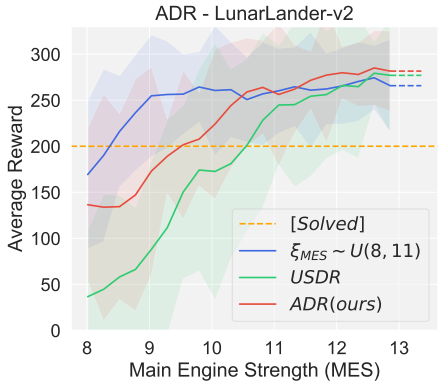


Figure 4. Generalization on `LunarLander-v2` for an expert interval selection, ADR, and USDR. Higher is better.

In Figure 6 explains the flexibility of ADR by showing generalization and sampling distribution at various stages of training. ADR starts by sampling approximately uniform for the first 650K steps, when it is apparent that it finds a deficiency in the policy on higher ranges of the MES. As those areas become more frequently sample between steps 650K-800K steps, the agent learns to solve all of the higher-MES environments, as shown by the generalization curve for 800K steps. As a result, the discriminator is no longer able to differentiate reference and randomized trajectories from the higher MES regions, and starts to reward environment instances generated in the lower end of the range ($\xi_{MES} \sim U[8, 11]$), which improves generalization by the completion of training.

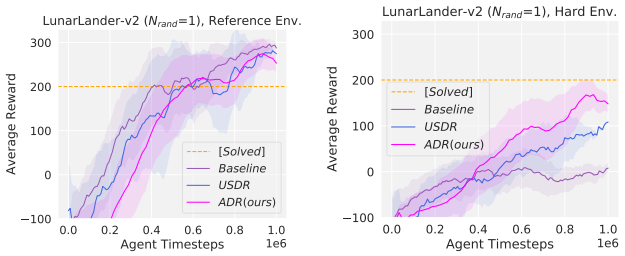
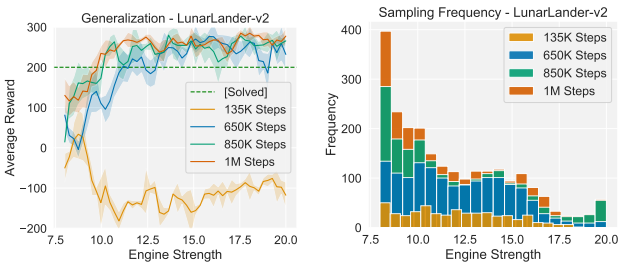


Figure 5. Learning curves for various agents on `LunarLander-v2`. Higher is better.

4.3. Randomization in High Dimensions

If the intuitions that drive ADR are correct, we should see increased benefit of a learned sampling strategy with larger N_{rand} (degree of randomization space), due to the increasing sparsity of *informative* environments. We first explore ADR's performance on `Pusher3DOF-v0`, an environment where $N_{rand} = 2$. Both randomization dimensions (puck damping, puck friction loss) affect whether or not the puck retains momentum and continues to slide after making



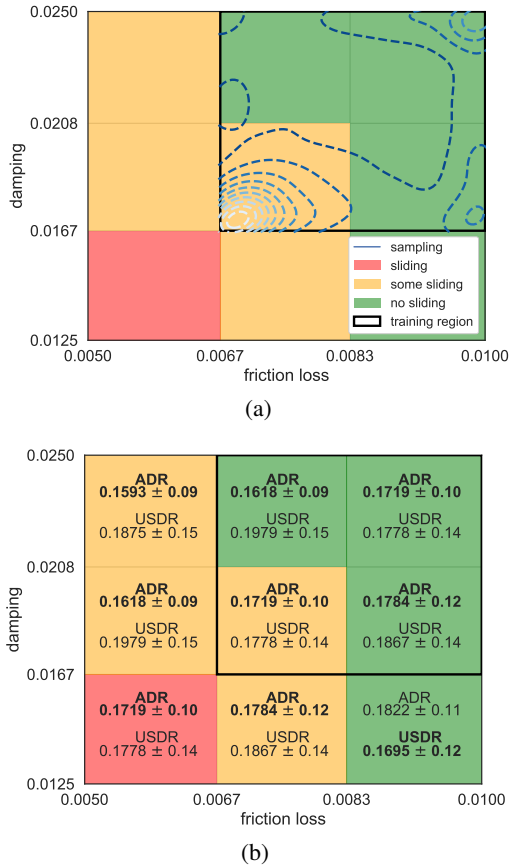
(a) (b)

Figure 6. Agent generalization (a) and environment sampling frequency (b) throughout training on `LunarLander-v2`.

contact with the agent's end effector. Lowering the values of these parameters *simultaneously* creates an intuitively-harder environment, where the puck slides more and more after being hit. In the reference environment, the puck retains no momentum and must be continuously pushed in order to move. We qualitatively visualize the effect of these parameters on puck sliding in Figure 7(a).

From Figure 7(b), we start to see ADR's robustness to *extrapolation* - or when the target domain lies *outside the training region*. We train two agents, one using ADR and one using USDR, and show them only the training regions encapsulated by the dark, outlined box in the top-right of Figure 7(a). Qualitatively, only 25% of the environments have dynamics which cause the puck to slide, which are the hardest environments to solve in the training region. We see that from the sampling histogram overlaid on Figure 7(a) that ADR prioritizes the single, harder orange region more than the green regions, allowing for better generalization to the unseen test domains, as shown in Figure 7(b). Unsurprisingly, ADR outperforms USDR in all but one test region, which is a region where the sliding dynamics of the puck are insignificant. In general, we also see that ADR produces policies with less total variance than their USDR counterparts.

From both Figure 8(a) and Figure 8(b), which are learning curves for USDR and ADR on the reference environment and hard environment (the red square in Figure 7) respectively, we observe an interesting phenomenon: not only does ADR solve both environments more consistently (i.e doesn't pop up above the *Solved* line), but USDR also *unlearns* the good behaviors it acquired in the beginning of training. When training Deep Neural Networks in both supervised and reinforcement learning settings this phenomenon has been dubbed as *catastrophic forgetting* (Kirkpatrick et al., 2016). ADR seems to exhibit this slightly (leading to "hills" in the curve), but due to the adaptive nature the algorithm, it is able to adjust more quickly and retain solid performance across all environments.



(a) (b)

Figure 7. Agent generalization (b) and environment sampling frequency (a) throughout training on `Pusher3DOF-v0`.

4.4. Randomization in *Uninterpretable* Dimensions

We further show the significance of ADR over traditional DR on `ErgoReacher-v0`, where $N_{rand} = 8$. The randomized parameters, joint gains and max torques for each joint, lose all intuition of which environments are *hard* due to the complex interactions between the parameters. For demonstration purposes, we again test extrapolation by creating a held-out target environment with extremely low values for torque and gain, which means that certain states in the environment can cause *catastrophic* failure - i.e gravity pulls the robot end effector down, and the robot is not strong enough to pull itself back up. We show an example of an agent getting trapped in such a catastrophic failure state in Figure 9.

To generalize effectively, the robot should prioritize environments with lower torque and gain values in order to learn how to operate in such states precisely. However, since the hard evaluation environment is not seen during training, ADR must learn to prioritize the hardest environments that it can see, and learn behaviors that *can* operate well across the entire training region.

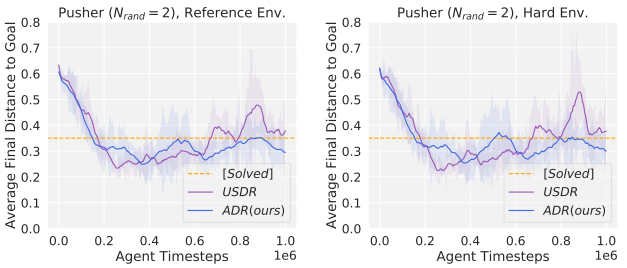


Figure 8. Final distance from goal on a reference and held-out environment in `Pusher-3DOF-v0`, plotted over time. Lower is better.

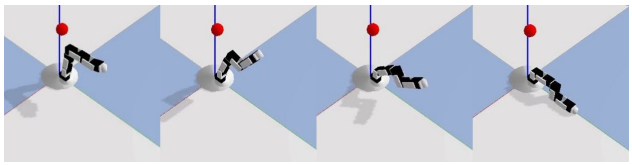


Figure 9. An example progression (left to right) of an agent moving to a catastrophic failure state (Panel 4) in the hard `ErgoReacher-v0` environment.

In Figure 10(a), we see that when evaluated on the reference environment, the policy learned using ADR has much lower variance than one learned using USDR and can solve the environment much more effectively. In addition, it generalizes better to the unseen target domain as shown in Figure 10(b), again which much less variance in the learned agent policy.

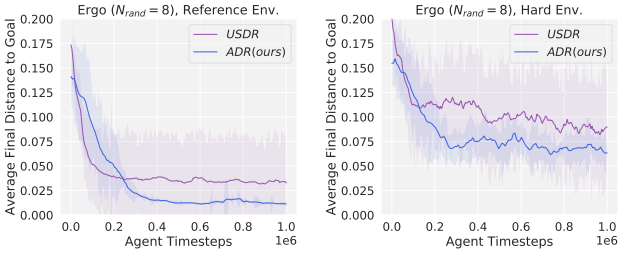


Figure 10. Final distance from goal on a reference and held-out environment in `ErgoReacher-v0`, plotted over time. Lower is better.

USDR's high variance on `ErgoReacher-v0` is indicative of some of the its issues: by continuously training on a random mix of hard and easy MDP instances, agent behaviors, both beneficial and detrimental, can be learned and unlearned throughout training, leading to both inconsistent and unpredictable behavior upon transfer. By focusing on those harder environments and allowing the definition of *hard* to adapt over time, ADR shows more consistent performance and better overall generalization than DR in all environments tested.

4.5. Bootstrapping Training of New Agents

Unlike DR, ADR's learned sampling strategy and discriminator can be reused to train new agents from scratch. To test the transferability of the sampling strategy, we first train an instance of ADR on `LunarLander-v2`, and then extract and freeze the SVPG particles and discriminator. We then replace the agent policy with an random network initialization, and once again train according the the details in Section 4.1. From Figure 11(a), it can be seen that the bootstrapped agent generalization is competitive to the one learned with ADR from scratch. However, its training speed is relatively lower.

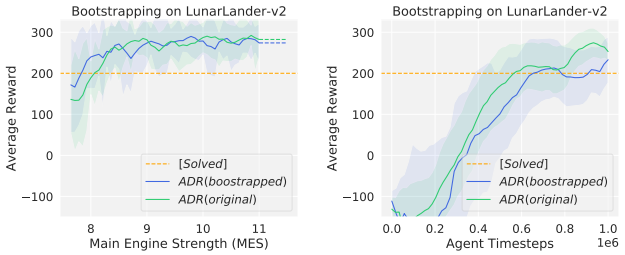


Figure 11. Generalization and learning progression on `LunarLander-v2` when using ADR to bootstrap a new policy. Higher is better.

4.6. Interpretability

One of the secondary benefits of ADR is its ability to give insight into incompatibilities between the simulation and randomization ranges. We show the simple effects of this in a one-dimensional `LunarLander-v2`, where we only randomize the main engine strength. Our initial experiments varied this parameter between 6 and 20, which lead to ADR learning degenerate agent policies by learning to propose the lopsided distribution in Figure 12. Upon inspection of the simulation, we see that when the parameter has a value of less than approximately 8, the task becomes almost impossible to solve due to the other environment factors (i.e the lander always hits the ground too fast).

After adjusting the parameter ranges to more sensible values, we see a better sampled distribution in yellow, which still gives more preference to the hard environments in the lower engine strength range. Most importantly, ADR allows for analysis that is both *focused* - we know exactly what part of the simulation is causing trouble - and *pre-transfer*, or done before a more expensive experiment such as real robot transfer has taken place. With USDR, the agents would be

equally trained on these degenerate environments, leading to agents that learn potentially undefined behavior in these truly out-of-distribution environments.

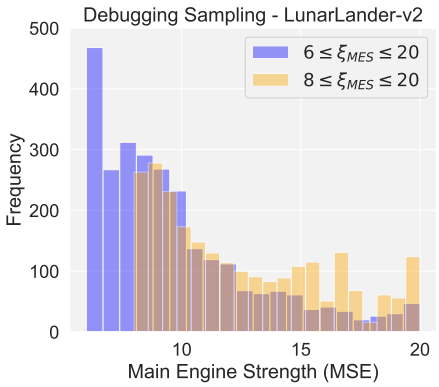


Figure 12. Sampling frequency across engine strengths when varying the randomization ranges. The updated, yellow distribution shows a much milder uneveness in the distribution, while still learning to focus on the harder instances.

5. Related Work

5.1. Simulation Randomization

Randomizing simulation parameters for better generalization or transfer performance is a well-established idea in evolutionary robotics (Zagal et al., 2004; Bongard & Lipson, 2004), but recently has emerged as an effective way to perform zero-shot transfer of deep reinforcement learning policies in difficult tasks (OpenAI et al., 2018; Tobin et al., 2017; Chebotar et al., 2018; Peng et al., 2018).

5.2. Dynamic and Adversarial Simulators

Dynamic and learnable simulations are an effective way to adapt a simulation to a particular target environment. Chebotar et al. (2018) and Ruiz et al. (2018) use reinforcement learning for effective transfer by learning parameters of a simulation that accurately describe the target domain, but require the target domain for reward calculation, which can lead to overfitting. Our approach requires no target domain, but rather only a reference domain (the default simulation parameters) and a general range of each parameter. ADR encourages diversity, and as a result gives the agent a wider variety of experience. In addition, unlike Chebotar et al. (2018), our method requires no carefully-tuned (co-)variances or task-specific cost functions. Concurrently, Khirodkar et al. (2018) also showed the advantages of learning adversarial simulations and disadvantages of purely uniform randomization distributions in object detection tasks.

To improve policy robustness, Pinto et al. (2017) (RARL) jointly trains both an agent and an adversary, who is applies environment forces that disrupt the agent's task progress. ADR removes the zero-sum game dynamics, which have been known to decrease training stability (Mescheder et al., 2018). More importantly, our method's final outputs - the SVPG-based sampling strategy and discriminator - are reusable and can be used to train new agents as shown in Section 4.5, whereas a trained RARL adversary would overpower any new agent and impede learning progress.

5.3. Informative Samples

Recently, Toneva et al. (2018) showed that certain examples in popular computer vision datasets are harder for networks to learn, and that some examples generalize (or are forgotten) much quicker than others. We explore the same phenomenon in the space of MDPs defined by our randomization ranges, and try to find the "examples" that cause our agent the most trouble. Unlike Toneva et al. (2018), we have no supervisory loss signal in RL, and instead attempt to learn a proxy signal via a discriminator.

5.4. Generalization in Reinforcement Learning

Generalization in RL has long been one of the holy grails of the field, and recent work like Packer et al. (2018) and Farebrother et al. (2018) highlight the tendency of deep RL policies to overfit to details of the training environment, but offer no real solution to this issue. Our experiments exhibit the same phenomena, but our method goes one step further by explicitly searching for and varying the environment aspects that our agent policy may have overfit to. We find that our agents, when trained on more frequently on these problematic samples, show better generalization while increasing sample efficiency and interpretability in the simulation's and agent policy's weaknesses.

6. Conclusion

In this work, we highlight failure cases of traditional domain randomization, and propose active domain randomization (ADR), a general method capable of finding the most informative parts of the randomization parameter space for a reinforcement learning agent to train on. ADR does this by posing the search as a reinforcement learning problem, and optimizes for the most informative environments using a learned reward and multiple policies. We show on a wide variety of simulated environments that this method efficiently trains agents with better generalization than traditional domain randomization, and extends well to high dimensional parameter spaces. We demonstrate the method's reusability and interpretability in such spaces, which can be used to train new agents faster and find more intuitive parameter ranges.

References

Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Bongard, J. and Lipson, H. Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pp. 57–62, 2004.

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

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., and Fox, D. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018.

Coumans, E. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3634-5. doi: 10.1145/2776880.2792704. URL http://doi.acm.org/10.1145/2776880.2792704.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Farebrother, J., Machado, M. C., and Bowling, M. Generalization and regularization in dqn, 2018.

Gangwani, T., Liu, Q., and Peng, J. Learning self-imitating diverse policies. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyxzRsR9Y7.

Golemo, F., Taiga, A. A., Courville, A., and Oudeyer, P.-Y. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pp. 817–828, 2018.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1803.06773*, 2018.

Khirodkar, R., Yoo, D., and Kitani, K. M. Vadra: Visual adversarial domain randomization and augmentation. *arXiv preprint arXiv:1812.00491*, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, Q. and Wang, D. Stein variational gradient descent: A general purpose bayesian inference algorithm. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2378–2386. Curran Associates, Inc., 2016.

Liu, Y., Ramachandran, P., Liu, Q., and Peng, J. Stein variational policy gradient, 2017.

Mescheder, L., Geiger, A., and Nowozin, S. Which training methods for gans do actually converge? In *International Conference on Machine Learning*, pp. 3478–3487, 2018.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

OpenAI, :, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. Learning dexterous in-hand manipulation, 2018.

Packer, C., Gao, K., Kos, J., Krhenbhl, P., Koltun, V., and Song, D. Assessing generalization in deep reinforcement learning, 2018.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018. doi: 10.1109/icra.2018.8460528. URL http://dx.doi.org/10.1109/ICRA.2018.8460528.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning, 2017.

Ruiz, N., Schulter, S., and Chandraker, M. Learning to simulate, 2018.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT Press, 2018.

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

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12.

Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

Zagal, J. C., Ruiz-del Solar, J., and Vallejos, P. Back to reality: Crossing the reality gap in evolutionary robotics. *IFAC Proceedings Volumes*, 37(8):834–839, 2004.

Ziebart, B. D. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, CMU, 2010.