2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-7281-6212-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/IROS45743.2020.9341019

Learning Domain Randomization Distributions for Training Robust Locomotion Policies

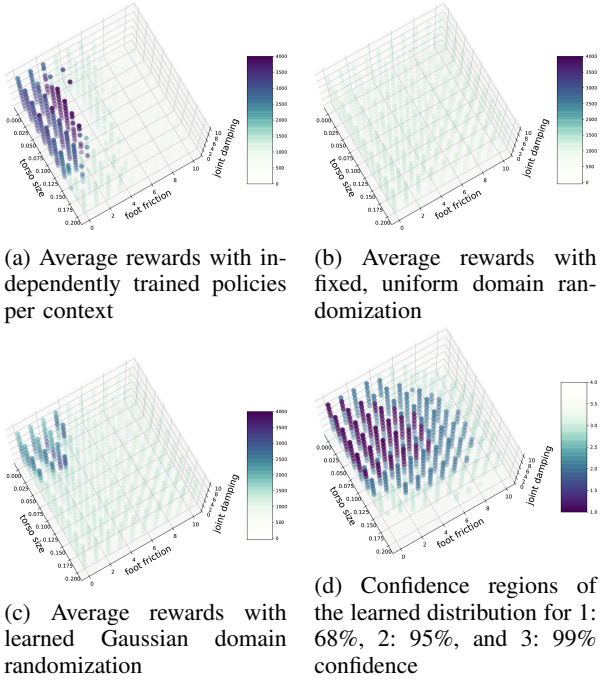
Melissa Mozifian*[1], Juan Camilo Gamboa Higuera*[1], David Meger[1] and Gregory Dudek[1]

***Abstract*— This paper considers the problem of learning behaviors in simulation without knowledge of the precise dynamical properties of the target robot platform(s). In this context, our learning goal is to mutually maximize task efficacy on each environment considered and generalization across the widest possible range of environmental conditions. The physical parameters of the simulator are modified by a component of our technique that learns the *Domain Randomization* (DR) that is appropriate at each learning epoch to maximally challenge the current behavior policy, without being overly challenging, which can hinder learning progress. This so-called sweet spot distribution is a selection of simulated domains with the following properties: 1) The trained policy should be successful in environments sampled from the domain randomization distribution; and 2) The DR distribution made as wide as possible, to increase variability in the environments. These properties aim to ensure the trajectories encountered in the target system are close to those observed during training, as existing methods in machine learning are better suited for interpolation than extrapolation. We show how adapting the DR distribution while training context-conditioned policies results in improvements on jump-start and asymptotic performance when transferring a learned policy to the target environment[1].**

I. Introduction

Learning robot behaviors in simulation is a important methodology as it reduces the need for potentially-unsafe state exploration with real hardware. A behavior-learning agent is, however, critically dependent on interaction with its environment. If no data is available for the real-world target robot (which might not even exist yet, in the robot factory use-case), can purely simulated learning be used to cover a wide range of potential physical robots? If so, how should the software simulations and learning modules interact? This paper proposes a method to learn behavior policies that generalize over many physical circumstances, by jointly learning behaviors and optimizing a *Domain Randomization* distribution for the software simulator. Our method achieves a balance between pushing the learner to increase its adaptation without becoming too challenging to prevent progress.

Except for simple robot systems in controlled environments, real robot experience may not correspond to situations that can be simulated; an issue known as the *reality gap* [1]. This gap limits the utility of simulation-based learning.



(a) Average rewards with independently trained policies per context

(b) Average rewards with fixed, uniform domain randomization

(c) Average rewards with learned Gaussian domain randomization

(d) Confidence regions of the learned distribution for 1: 68%, 2: 95%, and 3: 99% confidence

Fig. 1: Figure 1a shows the maximum rewards obtained by exhaustively running PPO on a grid of simulation parameters, 1b shows the performance of a policy trained with uniform domain randomization over the grid, 1c results from our method when learning the parameters of a Gaussian domain randomization distribution (whose confidence regions are shown in 1d). For fixed and learned DR the reward averages were obtained by evaluating the policy with 10 different rollouts.

Domain randomization (DR) aims to address the reality gap by training policies to maximize performance over a diverse set of simulation models, where the parameters of each model are sampled randomly from a given task distribution. The effectiveness of DR has been demonstrated by training controllers purely in simulation that subsequently transfer successfully to real robot systems [2], [3] and by fine-tuning the DR distributions with real world data [4].

While successful, an aspect that has not been addressed in depth is the selection of the DR distribution, which should be selected carefully to ensure the ability to generalize within the simulated experience. A DR distribution that is too wide may result in an under-performing policy (see Fig. 1b) or slow convergence (see Fig. 9). Conversely, training on a distribution that is too narrow leads to policies that do not generalize well [5], [6]. Most of the existing methods [5], [7], [8] assume the availability of real robot data. We study the extreme case where no real robot data is available, similar to [3], where the appropriate selection of a DR distribution

*Equal contribution. [1] Montreal Institute of Learning Algorithms (MILA), and the Mobile Robotics Lab (MRL) at the School of Computer Science, McGill University, Montreal, Canada. Correspondence to: Melissa Mozifian melissa.mozifian@mcgill.ca, Juan Camilo Gamboa Higuera gamboa@cim.mcgill.ca

[1]Our code is available at: https://github.com/melfm/lsdr

is crucial.

Our proposed method aims to find a compromise between variety and performance, optimizing the DR distribution to focus on environments where high rewards are likely, while regularizing the distribution parameters via a KL-divergence objective (explained in Sec. IV) and using more data from the worst performing simulators (see the experiments in Sec V-A). Our algorithm learns the domain randomization distribution while simultaneously optimizing the policy to maximize performance over the learned distribution. Figure 1c shows our results towards finding the *sweet-spot* distribution. We evaluate our method on a variety of control problems from the OpenAI Gym suite of benchmarks [9]. We find that our method is able to improve on the performance of fixed domain randomization. We demonstrate our model's robustness to initial simulator distribution parameters, showing that our method repeatably converges to similar domain randomization distributions across different experiments.

II. Related Work

Developing robust or generalizable controllers has a long and rich history beyond the scope of this paper [10], [11]. Caruana et al [12] was one of the first to explore the advantages of multi-task learning in a robotics context. Baxter et al [13] considered multi-task learning in a Bayesian/information-theoretic context. Packer et al [6] present an empirical study of generalization in Deep-RL, testing interpolation and extrapolation performance of state-of-the-art algorithms when varying simulation parameters. Our work extends this by providing results for the case when the training distribution parameters are learned and change during policy training. Chebotar et al [7] train policies on a distribution of simulators, whose parameters are fit to real-world data. Their algorithm alternates between optimizing the policy under the DR distribution and updating the DR distribution by minimizing the discrepancy between simulated and real world trajectories. In contrast, we aim to learn policies that maximize performance over a diverse distribution of environments where the task is feasible, as a way of minimizing the interactions with the real robot. Similarly, Ramos et al [8] adopt a likelihood-free inference method that computes the posterior distribution of simulation parameter using real-world trajectories. Rajeswaran et al [5] propose a related approach for learning robust policies over a distribution of simulator models, by optimizing the worst-case policy performance, using the CVaR objective [14]. Their method infers the distribution of simulation models by maximizing the likelihood of real-world trajectories. In Section V we combine our method with the CVaR objective to encourage diversity of the learned DR distribution.

Related to learning the DR distribution, Paul et al [15] propose an algorithm to maximize the probability of sampling simulators where the expected policy improvement is maximized, using a Bayesian Optimization (BO) framework. Mehta et al [16] direct the policy search towards harder instances given the DR ranges. Their algorithm updates the DR distribution to produce simulated environments within which the learned policy behaves differently than a policy trained in a given reference environment. Close in spirit to our method, [17] proposed to automatically generate a distribution over randomized simulated environments by starting a very narrow, solvable DR distribution and incrementally widening this distribution as training progresses. Their strategy for learning the distribution, while simple, does not scale as the dimension of DR variables increases.

Other related methods [4], [18] rely on policies conditioned on *context*: variables used to represent unobserved physical properties (i.e. friction, mass) that can be varied in the simulator, either explicitly or implicitly. When explicit, the context is equal to the simulator parameters. When implicit, the mapping between context vectors and simulator environments is learned during training. At test time, when the true context is unknown and the policy is fixed, the context vector is optimized to maximize the task performance objective either by gradient descent [4] or population-based gradient-free search [18]. Our method follows a similar approach, but we focus on learning the DR distribution. Rakelly et al [19] use context-conditioned policies where the context is implicitly encoded. During training, their algorithm jointly learns the context-conditioned policy and a probabilistic mapping from trajectory data to context vectors. The learned mapping is used for online inference of the context vector. This is similar to the Universal Policies with Online System Identification method [20], which instead uses deterministic context inference with an explicit context encoding. These methods use a fixed DR distribution and could benefit from adapting it during training, as we propose in this work.

III. Problem Statement

We consider parametric Markov Decision Processes (MDPs) [21]. An MDP $\mathcal{M}$ is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0 \rangle$, where $\mathcal{S}$ is the set of possible states and $\mathcal{A}$ is the set of actions, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, encodes the state transition dynamics, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the task-dependent reward function, γ is a discount factor, and $\rho_0 : \mathcal{S} \rightarrow \mathbb{R}$ is the initial state distribution. Let s_t and a_t be the state and action taken at time t. At the beginning of each episode, $s_0 \sim \rho_0(.)$. Trajectories τ are obtained by iteratively sampling actions using the current policy, π, $a_t \sim \pi(a_t|s_t)$ and evaluating next states according to the transition dynamics $s_{t+1} \sim p(s_{t+1}|s_t, a_t, z)$, where z are the parameters of the dynamics. Given an MDP $\mathcal{M}$, the goal is then to learn policy π to maximize the expected sum of rewards $J_{\mathcal{M}}(\pi) = \mathbb{E}_\tau [R(\tau)|\pi] = \mathbb{E}_\tau [\sum_{t=0}^{\infty} \gamma^t r_t]$, where $r_t = r(s_t, a_t)$. In our work, we aim to maximize performance over a *distribution* of MDPs, each described by a *context* vector z representing the variables that change over the distribution: changes in transition dynamics, rewards, initial state distribution, etc. Thus, our objective is to maximize $\mathbb{E}_{z \sim p(z)} [J_{\mathcal{M}_z}(\pi)]$, where $p(z)$ is the domain randomization distribution. Similar to [18], [4], [19], we condition the policy on the context vector, $\pi(a_t|s_t, z)$. In our experiments, we let z encode the parameters of the transition model in a

Algorithm 1 Learning the policy and training distribution

Require: testing distribution $p(z)$, initial parameters of the learned distribution ϕ, initial policy π, buffer size B, total iterations N

```
for i ∈ {1, ..., N} do
    z ~ p_φ(z)
    s_0 ~ ρ_0(s)
    B = {}
    while |B| < B do
        a_t ~ π(a_t|s_t, z)
        s_{t+1}, r_t ~ p(s_{t+1}, r_t|s_t, a_t, z)
        append (s_t, z_k, a_t, r_t, s_{t+1}) to B
        if s' is terminal then
            z ~ p_φ(z)
            s_{t+1} ~ ρ_0(s)
        end if
        s_t ← s_{t+1}
    end while
    φ ← UpdateDistribution(φ, p(z), π)
    π ← UpdatePolicy(π, B)
end for
```

physically based simulator; e.g. mass, friction or damping.

IV. Proposed Method

We introduce **LSDR** (Learning the Sweet-spot Distribution Range) algorithm for concurrently learning a domain randomization distribution and a robust policy that maximizes performance over it. **LSDR** requires a prior distribution $p(z)$, which is a guess of what the domain randomization ranges should be. Instead of directly sampling from $p(z)$, we use a surrogate distribution $p_\phi(z)$, with trainable parameters ϕ.

Our goal is to find appropriate parameters ϕ to optimize $\pi(\cdot|s, z)$. **LSDR** proceeds by updating the policy with data conditioned on $z \sim p_\phi(z)$, and updating the ϕ based on the performance of the policy. This interleaved process of optimizing the policy and domain randomization distribution, yields a robust policy that uses the full capability of the simulator by training on the feasible regions of the simulated dynamics. We use a regularizer that encourages $p_\phi(z)$ to produce diverse samples, avoiding the situations depicted in Fig. 3. The idea is to sample more data from environments where improvement of the policy is possible, without collapsing to environments that are trivial to solve. We summarize our training and testing procedure in Algorithm 1 and 2. In our experiments, we use Proximal Policy Optimization (PPO) [22] for the UpdatePolicy procedure in Algorithm (1).

A. Learning the Sweet-spot Distribution Range

The goal of our method is to find a training distribution $p_\phi(z)$ to maximize the expected reward of the policy over $p(z)$, while gradually reducing the sampling frequency of environments where the task is not solvable[2]. Such a situation is common in physics-based simulations, where a suboptimal selection of simulation parameters may lead to environments where the task is impossible due to physical limits or unstable simulations. Given a prior distribution $p(z)$, for updating the training distribution, we use an objective of the following form

$$\arg\max_{\phi} \mathcal{L}_{DR}(\phi) - \alpha D_{KL}(p(z)||p_\phi(z)) \tag{1}$$

where the first term encourages improvement on environments that are more likely to be solvable, while the second term keeps the distribution from collapsing. In our experiments, we set $\mathcal{L}_{DR}(\phi) = \mathbb{E}_{z\sim p(z)}[J_{\mathcal{M}_z}(\pi)\log(p_\phi(z))]$ so that the distribution is driven towards high reward regions. If we use the performance of the policy alone to determine whether the task is solvable for a given context z, then a trivial solution would be to make $p_\phi(z)$ collapse to a few easy environments. The second term in Eq. (1) helps to avoid this issue by penalizing distributions that deviate too much from $p(z)$, which is assumed to be wide. To estimate the gradient of Eq. (1) with respect to ϕ, we use the log-derivative score function gradient estimator [23], resulting in the following Monte-Carlo update :

$$\phi \leftarrow \phi + \lambda\Bigg[\frac{1}{K}\sum_{i=1}^{K}\Big(J_{\mathcal{M}_{z_i}}(\pi)\nabla_\phi \log(p_\phi(z_i))\Big) - \alpha\nabla_\phi D_{KL}(p(z)||p_\phi(z))\Bigg] \tag{2}$$

where $z_i \sim p(z)$. Updating ϕ with samples from $p_\phi(z)$, can be problematic as we may not get information on the policy performance in low probability contexts under $p_\phi(z)$.

We observed this issue in our single dimensional experiments with discrete distributions, where the distribution collapses as it never sees low-probability samples (see Fig. 3). An option in that case would be to use samples from $p(z)$ to evaluate the first term of our objective, which biases the gradients but avoids the distribution collapse[3]. To ensure that the two terms in Eq. (1) have similar scale, we standardize the evaluations of $J_{\mathcal{M}_{z_i}}$ with exponentially averaged batch statistics and set α via hyper-parameter search.

Algorithm 2 UpdateDistribution

Require: learned distribution parameters ϕ, testing distribution $p(z)$, policy π, total iterations M, total trajectory samples K

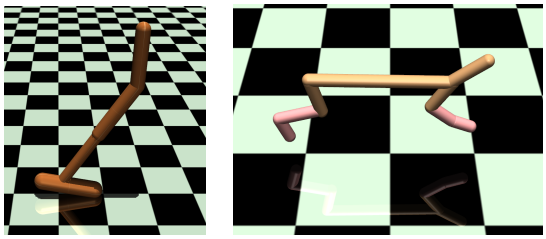
```
for i ∈ {1, ..., M} do
    sample z_{1:K} from p(z)
    Obtain Monte-Carlo estimate of L_DR(φ) by executing
    π on environments with z_{1:K}
    φ ← φ + λ∇_φ(L_DR(φ) − αD_KL(p(z)||p_φ(z)))
end for
```

V. Experimental Results

We evaluate the impact of learning the DR distribution on two standard benchmark locomotion tasks: Hopper and

[2]In this work, we consider a task solvable if there exists a policy that brings the environment to a set of desired goal states.

[3]For the experiments with multivariate Gaussian distributions changing the sampling distribution was not required.



(a) (b)

Fig. 2: Illustrations of the 2D simulated robot models used in the experiments. The hopper (a) and half-cheetah (b) tasks, present more challenging environments when varying dynamics.

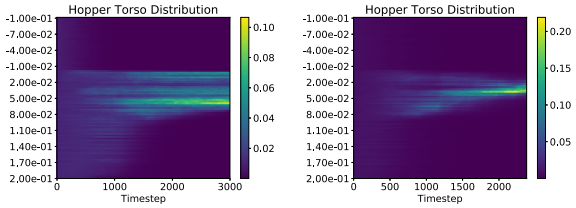
Half-cheetah, see Fig. 2, from the MuJoCo tasks in the OpenAI Gym suite [9]. We use an explicit encoding of the context vector z, corresponding to the torso size, density, foot friction and joint damping of the environments. Our experiments study both single-dimensional as well as multi-dimensional DR. For the first, we use discrete distributions and run experiments for each context dimension separately. For the latter, we parameterize the domain randomization distribution with multivariate Gaussians. We selected the prior $p(z)$ as a uniform distribution over ranges that include both solvable and unsolvable environments. In the case of discrete distributions, we discretize the support of $p(z)$ with 100 bins, and initialize $p_\phi(z)$ to be uniform; i.e. equivalent to $p(z)$. When sampling from this distribution, we first select a bin according to the discrete probabilities, then select a continuous context value uniformly at random from the ranges of the corresponding bin. For the multidimensional context experiments, the mean of the Gaussian distribution is initialized with the mean of test range, and the variance is initialized to $1/10^{th}$ of the variance of $p(z)$, so that most of the mass of the distribution falls within its support.

We compare the test-time jump-start and asymptotic performance of policies learned with $p_\phi(z)$ (learned domain randomization) and $p(z)$ (fixed domain randomization). At test time, we sample (uniformly at random) a test set of samples from the support of $p(z)$, 50 for single-dimensional and 100 for multidimensional contexts, and fine-tune the policy with the parameters obtained at training time. The questions we aim to answer with our experiments are: 1) Does learning policies with wide DR distributions affect the performance of the policy in the environments where the task is solvable? 2) Does learning the DR distribution converge to the actual ranges where the task is solvable? 3) Is learning the DR distribution beneficial?

A. Results

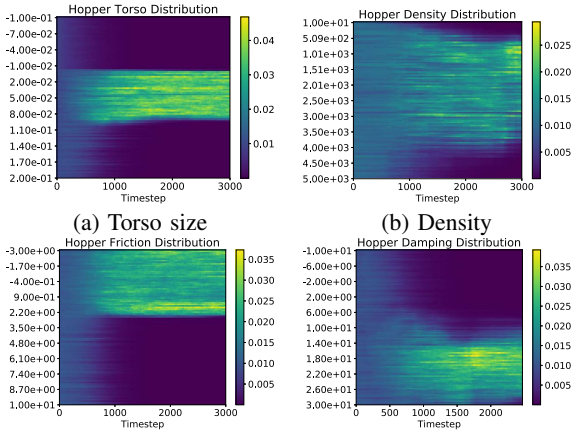
Learned Distribution Ranges: Table I shows the ranges for $p(z)$ and the final equivalent ranges for the distributions found by our method, for the single-dimensional discrete distribution learning experiments. Figures 4, 5, 6 and 7 show the evolution of $p_\phi(z)$ during training, using our method. In figures 4 and 5, each plot corresponds to a separate DR experiment, where we randomized one different simulator parameter while keeping the rest fixed.

Initially, each of these distributions is uniform. As the agent becomes better over the training distribution, it becomes easier to discriminate between promising environments (where the task is solvable) and impossible ones where rewards stay at low values. After around 1500 epochs, the distributions have converged. For Hopper, the learned distribution corresponds closely with the set of environments where we can find a policy using vanilla policy gradient methods from scratch. To determine the consistency of these results, we ran the Hopper torso size experiment 7 times, and fit the parameters of a uniform distribution to the resulting $p_\phi(z)$. The mean ranges ($\pm$ one standard deviation)



(a) (b)

Fig. 3: Learned torso size distribution for Hopper. Fig. 3a shows distribution learned without the KL divergence regularizer and Fig. 3b shows the distribution learned while sampling from train distribution.



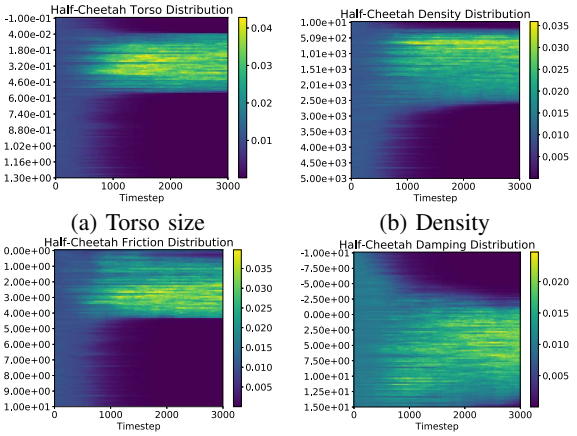
(a) Torso size (b) Density

(c) Friction (d) Damping

Fig. 4: Progress of **LSDR** for learning $p_\phi(z)$ for **Hopper**. Each plot corresponds to a different experiments where we kept the other simulator parameters fixed at their default values. Lighter color corresponds to higher probabilities.

Environment	Parameters	$p(z)$	Converged $p_\theta(z)$
Hopper	Torso size	$[-0.1, 0.2]$	$[0.0015, 0.09]$
	Density	$[10, 5000]$	$[400, 4000]$
	Friction	$[-3.0, 10.0]$	$[-3.0, 3.0]$
	Joint Damping	$[-10.0, 30.0]$	$[-1.0, 26]$
Half-Cheetah	Torso size	$[-0.1, 2.0]$	$[0.055, 0.52]$
	Density	$[10, 5000]$	$[285, 2420]$
	Friction	$[-3.0, 10.0]$	$[0.65, 4.39]$
	Joint Damping	$[-10.0, 30.0]$	$[-1.65, 12.9]$

TABLE I: Ranges of parameters for each environment, in the beginning of training and the equivalent ranges found by the algorithm, obtained by fitting an uniform distribution to the final learned distribution.



(a) Torso size (b) Density

(c) Friction (d) Damping

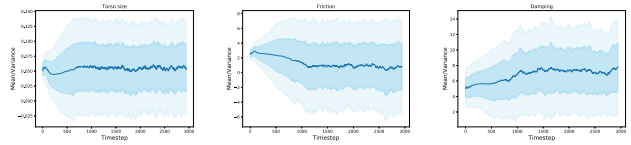
Fig. 5: Distribution evolution of **LSDR** for learning $p_\phi(z)$ for **Half-Cheetah**. Note how the learned distribution assigns low probability to physically implausible environments (negative or large mass, large friction etc.)

across the 7 experiments were $[0.00086 \pm 0.00159, 0.09275 \pm 0.00342]$, which provides some evidence for the reproducibility of our method. For figures 4 and 5, all plots correspond to a multidimensional context experiment. Since the contexts are being learned jointly, the final distributions correspond to the joint distribution of all contexts. These plots correspond to the 2D projection of each context distribution.

Learned vs Fixed Domain Randomization: Using PPO as the policy optimizer, we compare the performance difference between learning the domain randomization distribution (**LSDR**) and keeping it fixed (**Fixed-DR**). We ran the same experiments for *Hopper* and *Half-Cheetah*. Fig. 8 depicts learning curves when fine tuning the policy at test-time, for torso size randomization. All the methods start with the same random seed at training time across 100 seeds. The policies are trained for 3000 epochs, where we collect $B = 4000$ samples per epoch for the policy update. For the distribution update, we collect $K = 10$ additional trajectories and run the gradient update for $M = 10$ steps (without re-sampling new trajectories). To ensure fairness in terms of the sampled contexts, we report the comparison over the full test range (the support of $p(z)$). From these results, it is clear that **LSDR** improves on the jump-start and asymptotic performance over **Fixed-DR**. Note that by testing policy generalization on the full support of $p(z)$, our tests include contexts that are not solvable; i.e. contexts in which the optimal policy found by vanilla-PPO does not result in successful locomotion.[4].

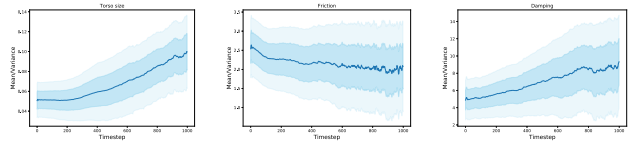
For the multidimensional Hopper experiments, the results are shown in Fig.1 1. In this case, we evaluated the performance of the policy at the last training iteration, with 10 rollouts per context. For the independently trained policies, we ran PPO for each context in the grid for 1000 epochs, each with 4000 steps of experience. While our method does not match the performance of the exhaustive grid

[4]Successful policies on Hopper obtain cumulative rewards of at least 2500. For Half-Cheetah, the rewards are greater than 0 when the robot successfully moves forward.



(a) Torso size (b) Friction (c) Damping

Fig. 6: Progress of **LSDR** for learning a multidimensional Gaussian $p_\phi(z)$ for $p_\phi(z)$ for **Hopper**.



(a) Torso size (b) Friction (c) Damping

Fig. 7: Progress of **LSDR** for learning a multidimensional Gaussian $p_\phi(z)$ for **Half-Cheetah**.

training, it is clearly better than training on an uniform DR distribution. Furthermore, our method observes considerably less experience data ($1.2e7$ steps) than the exhaustive grid training ($4e9$ steps). We expect to get better performance by increasing the per-epoch experience, as done in [20].

Using a different policy optimizer We also experimented with using EPOpt-PPO [5] as the policy optimizer in Algorithm (1). The motivation for this is to mitigate the bias towards environments with higher cumulative rewards $J_{M_z}(\pi)$ early during training. EPOpt encourages improving the policy on the worst performing environments, at the expense of collecting more data per epoch. At each training epoch, we obtain 100 samples from the training distribution and obtain trajectories by executing the policy once on each of the corresponding environments. From the resulting 100 trajectories, we use the 10% trajectories that resulted in the lowest rewards to fill the buffer for a PPO policy update, discarding the rest of the trajectories.

Fig. 9 compares the effect of learning the DR distribution vs. using a fixed wide range. We found that learning the DR distribution resulted in faster convergence to high reward policies over the evaluation range [0.01, 0.09]: our method attains cumulative rewards greater than 3000 in about one half of the epochs required with uniform DR. It also results in better asymptotic performance: it finds policies with better and more consistent performance (less variance) as seen in Figure 9. This may be a consequence of lower variance in the policy gradient estimates, as the the learned $p_\phi(z)$ has lower variance than $p(z)$. Interestingly, using EPOpt resulted in a distribution with a wider torso size range than vanilla PPO, from approximately 0.0 to 0.14, demonstrating that optimizing worst case performance helps alleviating the bias towards high reward environments.

VI. Discussion

By allowing the agent to learn a distribution of environments jointly with its behavior policy, the agent can learn to solve difficult control tasks over a wide range of physical parameters. Our main experimental validation is in the domain of simulated robotic locomotion. As shown in

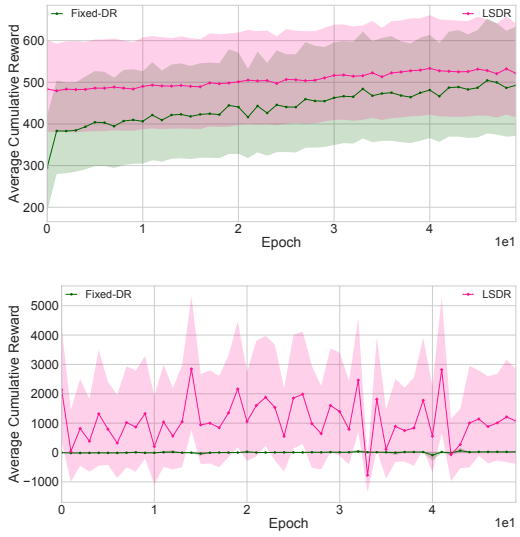


Fig. 8: Comparison of test-time performance on Hopper (top) and Half-Cheetah (bottom) between fixed vs learned domain randomization in the policy inputs. Note that running experiments on the test range includes environments that may not necessarily be solvable, increasing the variance on the performance across seeds.

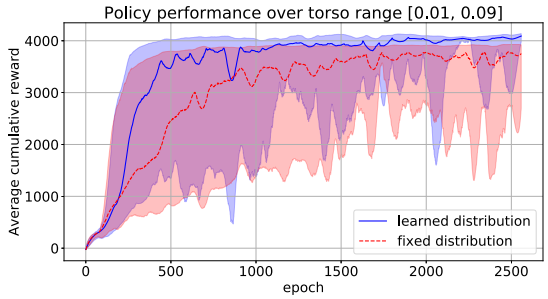


Fig. 9: Learning curves for torso DR on Hopper using EPOpt. Lines represent mean performance, while the shaded regions correspond to the maximum and minimum performance (smoothed with a 5th order Savitzky-Golay filter and a window of 10 epochs)

our experiments, our method is not sensitive to the initial domain randomization distribution and is able to converge to a diverse range, near to the limits of feasibility for the task. We experimented with single dimensional discrete and multivariate Gaussian distributions. Our results show the benefit of learning a domain randomization distribution when compared to the previously accepted practice: keeping it fixed. We learn policies that perform robustly across a wide range of environments, maximizing the learning progress in the context where no access to the true target robot is available. Further extension of this work will explore better objectives for learning the distribution and more flexible parameterizations for the DR distribution.

Unscaled rewards, non-linear dynamics and many other complex factors make the manual setting of DR ranges ill-suited for human intuition. Automating this step is crucial to improving the robustness of simulation-to-real transfer and solving today's wide range of robotics challenges; our method is a new step in this fruitful direction.

REFERENCES

[1] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*. Springer, 1995, pp. 704–720.

[2] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

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

[4] T. Chen, A. Murali, and A. Gupta, "Hardware conditioned policies for multi-robot transfer learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 9355–9366.

[5] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "EPOpt: Learning robust neural network policies using model ensembles," *CoRR*, vol. abs/1610.01283, 2016. [Online]. Available: http://arxiv.org/abs/1610.01283

[6] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing generalization in deep reinforcement learning," *arXiv preprint arXiv:1810.12282*, 2018.

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

[8] F. Ramos, R. Possas, and D. Fox, "Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators," in *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

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

[10] G. Zames, "Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses," *IEEE Transactions on Automatic Control*, vol. 26, no. 2, pp. 301–320, April 1981.

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

[12] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," in *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993, pp. 41–48.

[13] J. Baxter, "A bayesian/information theoretic model of learning to learn via multiple task sampling," *Machine Learning*, vol. 28, no. 1, pp. 7–39, Jul 1997.

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

[15] S. Paul, M. A. Osborne, and S. Whiteson, "Contextual policy optimisation," *CoRR*, vol. abs/1805.10662, 2018. [Online]. Available: http://arxiv.org/abs/1805.10662

[16] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," *CoRR*, vol. abs/1904.04762, 2019. [Online]. Available: http://arxiv.org/abs/1904.04762

[17] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," 2019.

[18] W. Yu, C. K. Liu, and G. Turk, "Policy transfer with strategy optimization," *CoRR*, vol. abs/1810.05751, 2018. [Online]. Available: http://arxiv.org/abs/1810.05751

[19] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," *arXiv preprint arXiv:1903.08254*, 2019.

[20] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[21] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2018.

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

[23] M. C. Fu, "Gradient estimation," *Handbooks in operations research and management science*, vol. 13, pp. 575–616, 2006.