Learning a System-ID Embedding Space for Domain Specialization with Deep Reinforcement Learning

James A. Preiss
Univ. of Southern California
Los Angeles, CA 90089
japreiss@usc.edu

Karol Hausman
Google Brain
Mountain View, CA 94043
karolhausman@google.com

Gaurav S. Sukhatme
Univ. of Southern California
Los Angeles, CA 90089
gaurav@usc.edu

Abstract

We propose a method to improve the ability of reinforcement-learned policies to adapt to unknown dynamics of the agent at test time. Our method learns an embedding space to distill system identification information into a form that is both useful for policy specialization, and easy to estimate.

1 Introduction and related work

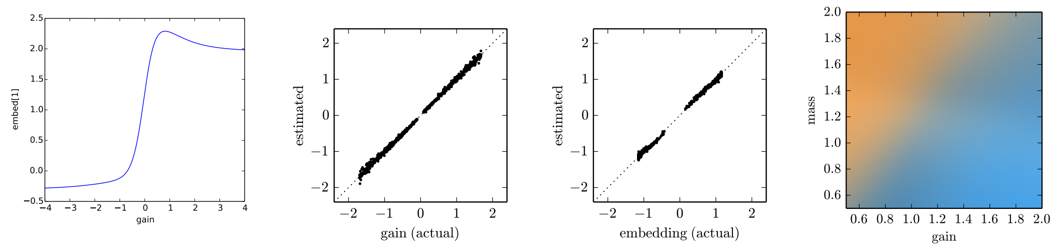
Policies trained with reinforcement learning (RL) are often brittle and fail to generalize beyond the training environment. An important instance of this problem is sim-to-real transfer for robotics. Domain randomization during training can improve robustness [1, 15, 13, 9, 11], but it is limited by its assumption that a single policy can perform adequately in all possible test domains. In addition, domain randomization introduces partial observability to the underlying MDP as the agent is not aware of the changes in its randomized dynamics. Meta-learning and fine-tuning [4, 8, 3] assume there will be an opportunity to collect data from the test domain and update the policy. In this work, we seek a policy that specializes to a novel test environment rapidly, without using significant data or computational effort at test time.

We assume that dynamics parameters are known during training but unknown under test. Our method includes two key contributions: 1) a learned embedding space that represents the system dynamics parameters in a form that is both useful and easy to identify, and 2) an observability reward that encourages the agent to maximize identification accuracy. We eliminate the task of estimating the actual dynamics parameters of the test environment, as in [14], without introducing the training complexity of a recurrent policy as in [7].

2 Problem statement and method

We consider RL in a continuous Markov decision process with states $\mathcal{S} \subseteq \mathbb{R}^n$, actions $\mathcal{A} \subseteq \mathbb{R}^d$, dynamics $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$, initial state distribution $\rho : \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$, and rewards $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. The state $\mathcal{S}$ is partitioned into an *observed space* $\mathcal{Z}$ consisting of the changing states such as angles and angular velocities of joints, and a *dynamics space* $\mathcal{D}$ consisting of joint geometries, moments of inertia, coefficients of friction, etc. During training, $d \in \mathcal{D}$ is known, but at test time the agent only observes $z \in \mathcal{Z}$. Our goal is to learn a policy π that maximizes a standard RL objective in expectation over a distribution $p_{\mathcal{D}}$ on $\mathcal{D}$. If $p_{\mathcal{D}}$ covers a wide enough range of system dynamics, it is reasonable to assume that no *blind* policy, where $\pi(a|z, d) = \pi(a|z) \; \forall \; d \in \mathcal{D}$, can perform adequately.

A natural approach is to train a policy $\pi(a|z, d)$ conditioned on the known value of d, alongside a *system identification* function that yields an estimate δ of d. At test time, we act with the policy $a \sim \pi(a|z, \delta)$. This method, explored by Yu et al. [14], creates two challenges. First, it requires estimating every dynamics parameter, even though some may be difficult to estimate, redundant, or unneeded to maximize reward. Second, behavior that maximizes reward in training may be suboptimal for system identification. It is preferable to learn a behavior that balances the primary reward with a secondary goal of making the system identification task as easy as possible.



(a) Learned mapping from gain g to one dimension of embedding space $\mathcal{E}$.

(b) Comparison of actual vs. estimated gain g (left) and embeddings $\mathcal{E}$ (right). Embedding spreads parameters requiring disjoint behavior to distant clusters.

(c) With redundant parameters, $g = m$ line (spatial) maps to same ε (color).

Figure 1: Desirable properties of learned embedding $e : \mathcal{D} \mapsto \mathcal{E}$ for point-mass environment.

We address both of these concerns by introducing a learned abstract representation $\mathcal{E} \subseteq \mathbb{R}^{d_\mathcal{E}}$ of the dynamics parameters. This creates an opportunity to distill the full complexity of $\mathcal{D}$ into a form that is potentially more useful for the policy to produce optimal behavior, while simultaneously being easier to estimate. During training, we learn an embedding function $e : \mathcal{D} \mapsto \mathcal{E}$ and a policy $\pi_\varepsilon(a|s, \varepsilon)$, where $\varepsilon = e(d)$. We simultaneously learn an identification function $id : (\mathcal{Z} \times \mathcal{A})^K \mapsto \mathcal{E}$, to estimate the embedding value from a fixed-length state-action trajectory. To aid observability, we augment the main RL reward with a term penalizing estimation error of id, thus rewarding behavior that makes estimating ε easier. Then, at test time, we store a rolling buffer τ of the past K pairs (z, a) and act with the policy $a_t \sim \pi_\varepsilon(a_t|z_t, id(\tau_{t-K:t}))$.

3 Experiments

In this section, we show desirable properties of our learned embedding space $\mathcal{E}$ and compare the performance of our architecture against several baselines. Details are given in Appendix B.

Point-Mass Environment: This low-dimensional system allows us to visualize learned embeddings. A 2D point mass $p \in \mathbb{R}^2$ follows dynamics $\ddot{p} = a$ for action $a \in [-1, 1]^2$ and reward $r = -\|p\|_2$. The system identification state d is a gain factor $g \in \pm[0.175, 1.75]$, such that the true force applied to the mass is $g \cdot a$. Due to the unknown sign of g, a domain randomization policy cannot possibly perform well in this environment. As shown in Figures 1a and 1b, the learned e "squashes" the gain factor g into positive and negative clusters, helping the policy switch between disjoint behavior styles.

To demonstrate distillation of redundant parameters, we construct a version of the point-mass environment with a mass parameter m alongside the gain g, such that $\ddot{p} = ga/m$, making all (g, m) combinations with the same ratio indistinguishable. This poses a problem for any approach that requires accurate identification of $\mathcal{D}$ at test time. In contrast, our framework learns an embedding where all (g, m) combinations with similar ratios map similar embedding values, as shown in Figure 1c.

Half-Cheetah Environment: As a more complex task, we demonstrate results on the *Half-Cheetah* planar locomotion environment from the OpenAI Gym [2]. We randomize 37 kinematic and dynamic parameters of the model (see Appendix B for details). The embedding space $\mathcal{E}$ is 8-dimensional; we did not observe significant sensitivity to this choice.

Results are shown in Figure 2. Variance of rewards is large due to the randomized dynamics. The *blind* policy fails to achieve high rewards because it is unable to specialize its behavior to the dynamics parameters. The *plain* policy, conditioned on d instead of ε, achieves similar training rewards but worse generalization error, and suffers negative rewards in some test environments, while *ours* does not.

In current experiments, we have not yet observed a significant effect of our observability reward term. In future work, we plan to investigate other systems where observability-seeking behavior is most critical.

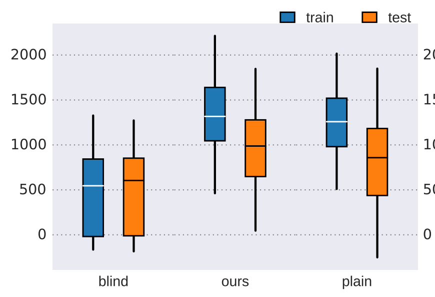


Figure 2: Box plot of training and test reward for *blind*, *ours*, and *plain* policies. Whiskers indicate full extent (min/max) of rewards.

References

[1] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *CoRR*, abs/1703.00472, 2017. URL `http://arxiv.org/abs/1703.00472`.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[3] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1126–1135, 2017.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 1856–1865. JMLR.org, 2018.

[6] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814. Omnipress, 2010.

[7] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017.

[8] Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *CoRL*, volume 78 of *Proceedings of Machine Learning Research*, pages 262–270. PMLR, 2017.

[9] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. 2017.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[11] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, 2017.

[12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[13] Jeroen van Baar, Alan Sullivan, Radu Cordorel, Devesh Jha, Diego Romeres, and Daniel Nikovski. Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics, 2018.

[14] Wenhao Yu, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. 2017.

[15] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei A. Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and imitation learning for diverse visuomotor skills. *CoRR*, abs/1802.09564, 2018.

A Expanded description of method

In this section, we give additional details on our proposed method outlined in Section 2. As previously mentioned, we operate in the continuous MDP defined by $(\mathcal{S}, \mathcal{A}, p, \rho, r)$. We partition the state space $\mathcal{S}$ into observed states $\mathcal{Z}$ and dynamics states $\mathcal{D}$. To aid exposition, we refer to the full state s and the pair (z, d) interchangeably as appropriate. The states in $\mathcal{D}$ influence the system dynamics, i.e.

$$d_1, d_2 \in \mathcal{D},\ d_1 \neq d_2 \implies p(s|a, z, d_1) \neq p(s|a, z, d_2) \tag{1}$$

for $s \in \mathcal{S},\ a \in \mathcal{A},\ z \in \mathcal{Z}$ in general, but d is unobserved by the agent at test time. This setting could also be formalized as a partially observable MDP (POMDP), but we use the present formalism to emphasize the following assumptions:

Assumption A.1. *The dynamics state d is independent of the observed states and actions:*

$$p(d_{t+1}|a_t, z_t, d_t) = p(d_{t+1}).$$

In this paper, we consider episodic RL and hold d constant during a single episode; however, our architecture does not prohibit training under non-stationary d and in non-episodic environments.

Assumption A.2. *The test-time observation is a deterministic truncated identity function $O(z, d) = z$, rather than a more general and/or stochastic function.*

This assumption emphasizes that we are not concerned with the partial observability resulting from, e.g., an agent perceiving its environment through a camera.

In training, the dynamics parameters follow the distribution $p_{\mathcal{D}}(d) : \mathcal{D} \mapsto \mathbb{R}_{\geq 0}$. We note that in practical applications, we are often free to choose $p_{\mathcal{D}}$, in which case we should choose a distribution that covers all values of d that might be encountered at test time. Our goal is to learn the embedding function $e : d \mapsto \varepsilon$ and a policy conditioned on the learned embedding $\pi_\varepsilon : \mathcal{Z} \times \mathcal{E} \times \mathcal{A} \mapsto \mathbb{R}_{\geq 0}$ that maximizes the expectation of a standard RL objective over $p_{\mathcal{D}}$:

$$\underset{\pi,\ e}{\text{maximize}}\ J_R(\pi_\varepsilon, e) \triangleq \mathbb{E}_{d \sim p_{\mathcal{D}},\ \tau \sim p(\tau|\pi_\varepsilon, e, d)} \sum_{t=0}^{H} r(s_t, a_t), \tag{2}$$

where $\tau = (s_0, a_0), \ldots, (s_H, a_H)$ denotes a state-action trajectory and $p(\tau|\pi_\varepsilon, e, d)$ denotes the trajectory distribution induced by π_ε and e under dynamics parameters d:

$$p(\tau|\pi_\varepsilon, d) = \rho(s_0) \prod_{t=0}^{H-1} p(s_{t+1}|s_t, a_t)\pi_\varepsilon(a_t|z_t, e(d_t)). \tag{3}$$

We simultaneously train the identification function id to minimize estimation error under the induced trajectory distribution:

$$J_{id}(id, \pi_\varepsilon, e) \triangleq \mathbb{E}_{d \sim p_{\mathcal{D}},\ \tau \sim p(\tau|\pi_\varepsilon, e, d)} \left[\sum_{t=0}^{H} \| id(\tau_{t-K:t}) - e(d) \|_2^2 \right], \tag{4}$$

While the embedding space may help make system identification easier, it has a failure mode that must be considered: if e and id both map to the same constant value for any input, id will achieve perfect performance during training, but the embedding will not influence the policy, and π_ε will be reduced to a *blind* policy equivalent to that learned under pure domain randomization. To avoid this problem, we impose a structure where the parameters of the embedding function e can only be learned via backpropagation of the RL objective gradient through π_ε. We then augment the main objective (2) with an *observability objective* term based on J_{id} to reward accurate system identification of ε. We additionally include an entropy regularization term,

$$J_{\mathcal{H}}(\pi_\varepsilon, e) \triangleq \mathbb{E}_{d \sim p_{\mathcal{D}},\ \tau \sim p(\tau|\pi_\varepsilon, e, d)} \mathcal{H}\left[\pi_\varepsilon(\cdot|z, e(d))\right], \tag{5}$$

because it is inherent in the Soft Actor-Critic (SAC) RL algorithm [5] we use in experiments (additional details in Appendix B). The complete RL maximization objective is given by

$$J(\pi_\varepsilon, e) = J_R(\pi_\varepsilon, e) - \alpha_{id} J_{id}(id, \pi, e) + \alpha_{\mathcal{H}} J_{\mathcal{H}}(\pi_\varepsilon, e). \tag{6}$$

The regularization weights $\alpha_{id} > 0,\ \alpha_{\mathcal{H}} > 0$ are user-chosen hyperparameters. Note that the summation in (4) intentionally includes negative values of the timestep t. At test time, id will be required to produce an input to π_ε before enough time has passed to fill the K-length history buffer. To include this case in training, we initialize the history buffer with zeros at the episode start (or clear it periodically during non-episodic training), and include a reward for these initial timesteps in (4). The value of states and actions for negative values of t are therefore defined as zero vectors.

We emphasize that the observability objective term (4) is added to the reinforcement learning reward rather than optimized directly. The parameters of id are considered constant when optimizing π_ε and e; and the parameters of e are considered constant when optimizing id. The system identification function id is not updated by the RL algorithm. It is updated in a separate supervised learning step, performed alongside each iteration of RL.

B Experimental details

B.1 Policy and identifier parameterizations

In our experiments, the policy π_ε is parameterized as a fully connected neural network with 2 hidden layers of 128 units each, using ReLU nonlinearities [6]. The same architecture is used to parameterize the value- and Q-function estimators used by SAC. The embedding function e network contains one hidden layer of 128 units.

We employ a one-dimensional convolutional architecture for the identifier function id, composed of three 1D-convolutional layers, each with 64 filters of width 3 and ReLU activation, followed by a single fully connected layer with 128 units, and a linear output layer. The convolutional architecture is chosen to allow a longer trajectory window K without requiring a large number of parameters ($K = 16$ in our experiments). Additionally, it matches the intuition that differentiation and/or integration of the state and action trajectories is often required to identify the underlying dynamics parameters. For example, an engineered approach for estimating the gain parameter of the point-mass environment could differentiate the velocity observation to obtain acceleration.

B.2 RL algorithm

For all experiments, we use the entropy-regularized Soft Actor-Critic (SAC) reinforcement learning algorithm [5]. SAC is an off-policy algorithm where a stochastic policy is trained only with TD-learned value function estimates. We observed significantly higher rewards using SAC compared to the on-policy, Monte Carlo policy gradient algorithm PPO [10]. We conjecture that the use of TD-learning and a replay buffer is especially helpful in our scenario compared to single-environment training, since the replay buffer helps prevent "forgetting" about areas of the dynamics distribution $p_{\mathcal{D}}$ that have not been sampled recently.

The value function estimators used by SAC are parameterized by fully-connected networks of identical structure to the policy. For *blind* policies, the value function networks are conditioned only on the observed state $z \in \mathcal{Z}$ to emulate domain randomization approaches. For *plain* policies, they are conditioned on both z and $d \in \mathcal{D}$ to emulate the setup of [14]. For *ours* policies, they are conditioned on z, d, and $\varepsilon \in \mathcal{E}$. The gradient update of the embedding function e is only taken w.r.t. the policy reward and not the value function training losses.

B.3 Randomized Half-Cheetah environment

Here we give more details on our procedure for randomizing the Half-Cheetah environment. The following parameters are randomized: lengths of seven kinematic links, damping, stiffness [1], and ranges of six planar revolute joints, and gear ratios of six rotary actuators. In total, 37 parameters are randomized. For joint ranges, limits on either side are shifted by $\omega \sim \mathrm{unif}(-0.3, 0.3)$ radians. All other parameters must take nonnegative values, and some should not be too close to zero. Thus, we multiply those parameters by a random nonnegative ratio β^p where $p \sim \mathrm{unif}(-1, 1)$. The parameter $\beta > 1$ controls the amount of randomness. For example, if $\beta = 2$, then $1/2 \leq \beta^p \leq 2$. In our

[1] Joint stiffness in MuJoCo denotes the strength of a virtual spring pulling the joint to its resting position.

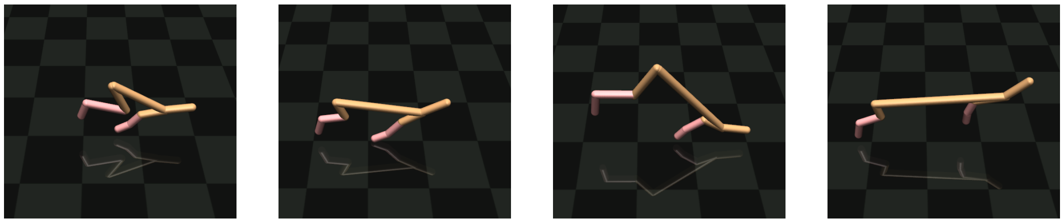


Figure 3: Variations of Half-Cheetah environment produced by randomization of kinematic and dynamic properties.

experiments, $\beta = 1.75$, which was the largest value we could use without generating too many overly difficult configurations. Some examples of the randomized half-cheetahs are shown in Figure 3.

Due to the architecture of the MuJoCo physics simulator used in this environment [12], it is not practical to sample new random dynamics parameters d for each training iteration. Instead, we construct a "universe" of 256 models initially, and sample a new "universe" at test time.