Simulation-Acquired Latent Action Spaces for Dynamics Generalization

Nicholas Corrado, Yuxiao Qu, Josiah P. Hanna
Department of Computer Sciences
University of Wisconsin – Madison
{ncorrado, qu45}@wisc.edu, jphanna@cs.wisc.edu

Abstract

Deep reinforcement learning has shown incredible promise at training high-performing agents to solve high-dimensional continuous control tasks in a particular training environment. However, to be useful in real-world settings, long-lived agents must perform well across a range of environmental conditions. Naively applying deep RL to a task where environment conditions may vary from episode to episode can be data inefficient. To address this inefficiency, we introduce a method that discovers structure in an agent's high-dimensional continuous action space to speed up learning across a range of environmental conditions. Whereas prior work on finding so-called latent action spaces requires expert demonstrations or on-task experience, we instead propose to discover the latent, lower-dimensional action space in a simulated environment and then transfer the learned action space for training over a distribution of environments. We evaluate our novel method on randomized variants of simulated MuJoCo environments and find that, when there is a lower-dimensional action space to exploit, our method significantly increases data efficiency. For instance, in the Ant environment, our method reduces the 8-dimensional action space to a 3-dimensional action space and doubles the median return achieved after a training budget of 2 million timesteps.

1 Introduction

Autonomous agents deployed in real-world settings often must generalize skills across a range of different environment dynamics. For instance, a quadruped robot that can walk perfectly on a smooth laboratory floor but cannot walk fluidly on pavement has limited real-world applicability. In this paper, we focus on the challenge of training agents to perform a single task over a distribution of environments with different physics parameters.

Deep reinforcement learning (RL) algorithms have demonstrated incredible success in learning difficult control tasks (Agostinelli et al., 2019; Andrychowicz et al., 2020; Schulman et al., 2016) and can be applied to learn over a distribution of environments simultaneously. However, in high-dimensional continuous control problems, RL requires a large number of environment interactions to achieve good performance even in a single environment. This data inefficiency often makes RL impractical in domains such as robotics, where training on a real robot is expensive and time consuming. Prior work suggests that data efficiency can be improved by learning a policy that takes actions in a lower-dimensional *latent action space*. The lower-dimensional latent actions are then mapped back to the original, higher-dimensional action space using a learned mapping. While a promising approach, finding a latent action space that can represent the task-optimal policy and increase data efficiency is difficult, possibly requiring advance knowledge of the optimal policy.

We note that in many tasks, we have access to a simulator in which learning the task-optimal policy is comparatively cheap. We hypothesize that a latent action space learned in such a simulator can increase data efficiency when training directly on a distribution of environments. While it may seem appealing to instead directly transfer a simulation-trained policy to the distribution of environments, this approach is undesirable for two reasons: (1) small discrepancies between simulator dynamics and dynamics within the environment distribution can cause simulation-trained policies to transfer poorly (Kober et al., 2013), and (2) even if we have access to a high-fidelity simulator, it may be unable to emulate all possible physics settings within the environment distribution. To obtain strong performance, we must eventually train in the environment distribution. If the optimal policies in the simulator and environment distribution have similar lower-dimensional action representations, then a simulation-acquired latent action space may improve learning over the distribution of environments.

In this paper, we introduce SALAS (**S**imulation-**A**cquired **L**atent **A**ction **S**paces), a transfer technique that discovers a latent action space using rollouts from a policy trained in a simulated source environment and then transfers this latent space for training in a target distribution of environments. SALAS has two key features: (1) a method for learning latent action spaces of any dimensionality and (2) a heuristic for selecting the latent space dimensionality that yields the greatest improvement in data efficiency without exhaustively training with each latent space.

To evaluate our proposed method, we consider three simulated source environments and create target environment distributions by randomizing their simulation parameters. We show that a latent action space learned in the source environment can speed up learning over the target environment distribution, whereas training an agent in its native action space can yield poor data efficiency. Furthermore, we evaluate the latent space selected by SALAS in hindsight, comparing empirical results against the heuristic's predictions.

We now summarize our primary contributions:

- We introduce SALAS, a method that discovers a lower-dimensional action space to speed up learning over a distribution of environments with different physics parameters.
- We empirically demonstrate that a latent action space produced by SALAS in a fixed simulator instance can improve data efficiency when training an RL agent over a distribution of similar environments.
- Our experiments uncover cases where action space reduction reduces data efficiency even when the latent action space can represent a high-performing policy. We hope these findings will guide future work in action space representation learning.

Overall, we find that SALAS can be an effective tool for training agents to perform well across a variety of environmental conditions.

2 Related Work

In this section, we provide a summary of the most relevant prior work in action representation learning, sim-to-real RL, and multi-task RL.

2.1 Latent Action Representations

Our work focuses on learning structure in the actions of an agent's policy. While we consider the continuous action setting, prior work has investigated learning representations for discrete action as well. Chandak et al. (2019) learn a continuous embedding for discrete actions based on how each action affects the state, allowing for generalization across similar actions that is otherwise not possible in the original discrete space, and Jain et al. (2020) learn a continuous embedding that can generalize to unseen actions. Tennenholtz & Mannor (2019) adopt existing methods in natural language processing to inject prior information into action representations.

Other prior work focuses on representations for continuous action spaces. Luck et al. (2014; 2016) learn a latent action space in an online fashion to generate correlated noise, allowing for better exploration in the original action space. Similar to our work, they discover a linear latent-to-native mapping using Probabilistic PCA, though our method learns the mapping offline and reduces the dimensionality of the action space. Allshire et al. (2021) use a state-conditioned variational autoencoder to learn a latent-to-native mapping. Losey et al. (2020) similarly use an autoencoder to learn a user-friendly lower-dimensional action space for the teleoperation of robots. While these works focused on learning latent-to-native action mapping for a single similar or identical task, our work seeks improvement over a distribution of similar environments. Furthermore, rather than treating the dimensionality of the latent action space as an additional hyperparameter to tune, SALAS provides a heuristic to identify an appropriate latent dimensionality.

2.2 Sim-to-Real Reinforcement Learning

Since simulators are often a faster, cheaper, and safer alternative to training in the real world, much attention has been given towards using simulated experience for faster real-world training (Koos et al., 2010; Cutler & How, 2015). However, discrepancies between simulator dynamics and real-world dynamics introduce a *reality gap* (Kober et al., 2013), often causing a simulation-trained agent to perform poorly in the real world. Sim-to-real transfer techniques aim to bridge this gap. Our proposed method can be interpreted as a sim-to-real transfer technique where a latent action space learned in simulation is transferred to the real world (*i.e.* the target environment distribution). Cutler et al. (2014) use multiple simulators of increasing fidelity to minimize the number of real-world samples needed for learning. Christiano et al. (2016) learn an inverse dynamics model to adapt simulation-trained policies to the real world. More closely related to our work, Cully et al. (2015) use simulation to learn low-dimensional behaviors to guide a robot to adapt its behavior if it becomes damaged in the real world. Instead of using dimensionality reduction to better explore the original action space, SALAS reduces the action space itself, excluding actions that a trained policy is unlikely to choose.

Rather than treating the simulation as a low-fidelity model of the real world, domain randomization (DR) techniques model the reality gap as variability in simulation features and/or dynamics. The key insight is that an agent trained to perform well over a distribution of simulations will likely generalize well in the real world. Several prior works have transferred vision-based policies without the use of real-world images by randomizing low-fidelity simulation rendering (Tobin et al., 2017; Sadeghi & Levine, 2017; James et al., 2017). Peng et al. (2018) randomize simulator dynamics to transfer robotic arm policies to the real world. Since sampling environment parameters uniformly at random may introduce unnecessary variance into learning, Mehta et al. (2019) and Chebotar et al. (2019) improve data efficiency of DR through an adaptive sampling of environment parameters. Our proposed work is distinct from DR; in DR, agents are trained over a distribution of environments and evaluated on a single environment (the real world), whereas in our work, we train and evaluate agents in a distribution of environments.

Other methods seek to improve simulator fidelity through system identification (Kolev & Todorov, 2015) or grounding (Hanna et al., 2021). These methods are orthogonal to our approach; system identification and grounding augment simulation-trained polices for better real-world performance, whereas SALAS uses simulation only to learn a latent action space. In principle, such methods could be used to improve the simulator before applying SALAS.

2.3 Multi-Task Reinforcement Learning

In multi-task RL, an agent aims to learn a set of similar tasks simultaneously by leveraging similarities across the task set. When training over a distribution of environments, every environment instance can be interpreted as a different task. The setting considered in our work can thus be treated as an instance of multi-task RL where only the state transition function varies between tasks. Examples of prior work in the multi-task setting include distillation-based approaches (Czarnecki et al., 2019; Teh et al., 2017; Ghosh et al., 2018) that share behavioral structure across tasks as well as network architectures for representation sharing across tasks (D'Eramo et al., 2019; Sodhani et al., 2021). More closely related to our work, Hausman et al. (2018) learn an continuous skill embedding space for a set of tasks and then learns a task-conditioned policy in this embedding space over a different set of tasks. Our setting differs in that we do not provide agents with information on the current environment parameters.

3 Preliminaries

In this section, we formalize our problem setting and introduce the concept of a latent action space and policy.

3.1 RL Background

We formalize the environments of interest as finite horizon Markov decision processes (MDPs) defined by $(\mathcal{S}, \mathcal{A}, P_\phi, r, d_0, \gamma)$ where $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^k$ denote the continuous state and action space, respectively, $P_\phi(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ parameterized by ϕ denotes the probability density of transitioning to a state $\boldsymbol{s}'$ after taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$, $r(\boldsymbol{s}, \boldsymbol{a})$ denotes the reward for taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$, d_0 denotes the initial state distribution, and $\gamma \in [0, 1)$ denotes the discount factor. We write h to denote the horizon. Though we consider a continuous state space, the method we introduce applies to discrete state spaces as well. In this work, ϕ represents simulator physics parameters governing the state transition function.

We use $\mathcal{M}(\phi)$ to denote an environment with physics parameters ϕ and define a distribution over physics parameters p_{physics}. The target environment distribution $p_\mathcal{M}$ denotes the distribution over environments induced by sampling $\phi \sim p_{\text{physics}}$ and then instantiating $\mathcal{M}(\phi)$. We consider deterministic policies $\pi_\theta : \mathcal{S} \to \mathcal{A}$ parameterized by θ mapping states to actions. We seek a policy that maximizes the expected sum of discounted reward over the target environment distribution $J(\theta) = \mathbb{E}_{\pi, \boldsymbol{s}_0 \sim d_0, \mathcal{M} \sim p_\mathcal{M}} \left[\sum_{t=0}^{h} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$. We additionally assume access to one environment instance in $p_\mathcal{M}$ with physics parameters ϕ_0 which we call the source environment $\mathcal{M}_{\text{source}} = \mathcal{M}(\phi_0)$.

3.2 Latent Action Spaces and Policies

In high-dimensional continuous control problems, the optimal policy's actions may exhibit correlation across action dimensions. For example, as a quadruped robot walks, one leg moves back as another leg moves forward. These correlations provide an opportunity to reduce the dimensionality of the action space without substantially affecting optimal policy performance. A smaller action space can be explored more efficiently, potentially speeding up learning.

We define a latent action space $\tilde{\mathcal{A}}_{\tilde{k}} \subseteq \mathcal{A}$ with dimensionality $\tilde{k}$ and henceforth refer to $\mathcal{A}$ as the native action space. We use $\tilde{\boldsymbol{a}} \in \tilde{\mathcal{A}}$ and $\boldsymbol{a} \in \mathcal{A}$ to represent actions in the latent and native action spaces, respectively. A *latent policy* $\tilde{\pi}_\theta$ is parameterized by a policy network $f_\theta : \mathcal{S} \to \tilde{\mathcal{A}}$ mapping states to latent actions and a fixed latent-to-native action mapping $l : \tilde{\mathcal{A}} \to \mathcal{A}$. The overall latent policy is the composition of l and f_θ, *i.e.* $\tilde{\pi}_\theta(\boldsymbol{s}) = l(f_\theta(\boldsymbol{s}))$. Figure 1 illustrates the interaction protocol for the latent policy. We refer to a latent agent that uses latent space $\tilde{\mathcal{A}}_{\tilde{k}}$ as a latent-$\tilde{k}$ agent. If $\tilde{\mathcal{A}} = \mathcal{A}$ and l is the identity mapping, the policy is simply $\pi_\theta(\boldsymbol{s}) = f_\theta(\boldsymbol{s})$ and we refer to it as a *native policy*. Note that a latent policy does not necessarily operate in a lower-dimensional action space, since we may have $\tilde{\mathcal{A}} = \mathcal{A}$ and $l(\tilde{\boldsymbol{a}})$ not equal to the identity. In this case, $\tilde{\pi}_\theta$ operates in a transformed action space of the same dimensionality.

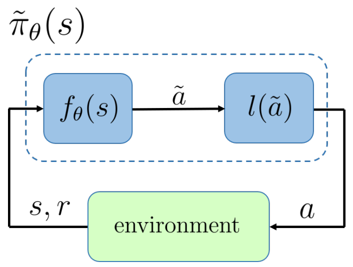


Figure 1: A latent policy $\tilde{\pi}_\theta(\boldsymbol{s}) = l(f_\theta(\boldsymbol{s}))$ with policy network $f_\theta(\boldsymbol{s})$ and latent-to-native mapping $l(\tilde{\boldsymbol{a}})$.

4 SALAS: Simulation-Acquired Latent Action Spaces

In this section, we introduce our primary algorithmic contribution: SALAS, a transfer technique that discovers a latent action space in $\mathcal{M}_{\text{source}}$ and then uses the latent space to increase data efficiency when training in $p_{\mathcal{M}}$. SALAS first trains a native agent to convergence in $\mathcal{M}_{\text{source}}$ and then forms a dataset of native actions using rollouts of the resulting policy. To obtain a set of latent action spaces of increasing dimensionality, SALAS then performs dimensionality reduction on this dataset. Using a heuristic, SALAS identifies which latent space yields the greatest improvement in data efficiency in $\mathcal{M}_{\text{source}}$. A new agent is then trained from scratch in $p_{\mathcal{M}}$ using the identified latent action space. A diagram illustrating the SALAS method can be found in Figure 2.

Our key insight is that if environments with support under $p_{\mathcal{M}}$ are sufficiently similar to $\mathcal{M}_{\text{source}}$, then the correlations across action dimensions in an optimal policy will be similar in both settings. If a latent action space learned in $\mathcal{M}_{\text{source}}$ improves data efficiency in $\mathcal{M}_{\text{source}}$, we expect that the latent space can also improve data efficiency in $p_{\mathcal{M}}$. A latent space may transfer poorly if the optimal policy action correlations for $p_{\mathcal{M}}$ are sufficiently different from $\mathcal{M}_{\text{source}}$. However, in practice, we observe that latent spaces can still improve data efficiency even we randomize physics parameters over relatively wide ranges.

4.1 Choosing a Latent Action Space Dimensionality

While dimensionality reduction of $\mathcal{A}$ may increase data efficiency, it may also prevent us from representing the task-optimal policy. It is unknown *a priori* how much we can reduce the dimensionalty of the action space to balance an increase in data efficiency with a possible decrease in final performance. Since we expect correlations across action dimensions in optimal policies for $\mathcal{M}_{\text{source}}$ and $p_{\mathcal{M}}$ to be similar, we evaluate latent spaces in $\mathcal{M}_{\text{source}}$ with the expectation that the most data efficient latent space in $\mathcal{M}_{\text{source}}$ will also be most data efficient in $p_{\mathcal{M}}$. While this approach avoids the data inefficiency associated with evaluating latent spaces in $p_{\mathcal{M}}$, it is still computationally expensive to identify the most data efficient latent space, especially if the native action dimension is large. Ideally, we

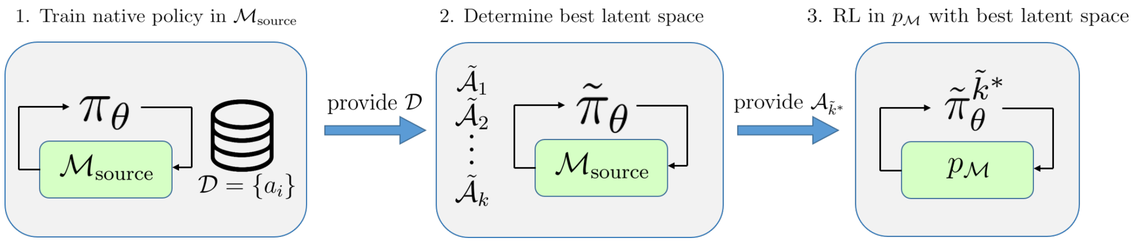

1. Train native policy in $\mathcal{M}_{\text{source}}$
π_θ
$\mathcal{M}_{\text{source}}$
$\mathcal{D} = \{a_i\}$
provide $\mathcal{D}$
2. Determine best latent space
$\tilde{\mathcal{A}}_1$
$\tilde{\mathcal{A}}_2$
$\tilde{\mathcal{A}}_k$
$\tilde{\pi}_\theta$
$\mathcal{M}_{\text{source}}$
provide $\mathcal{A}_{\tilde{k}^*}$
3. RL in $p_{\mathcal{M}}$ with best latent space
$\tilde{\pi}_\theta^{\tilde{k}^*}$
$p_{\mathcal{M}}$

Figure 2: An overview of SALAS. First, we train a native policy π_θ in the source environment and form an action dataset $\mathcal{D}$ using rollouts of the trained policy. Next, we constuct latent action spaces $\tilde{\mathcal{A}}_1, \ldots \tilde{\mathcal{A}}_k$ and use the heuristic described in Algorithm 1 to predict which latent space yields the best data efficiency. The heuristic involves training a small subset of latent agents in the source environment. After identifying the optimal latent dimensionality $\tilde{k}^*$, we train a latent-$\tilde{k}^*$ agent in the target environment distribution, denoted in the figure as $\pi_\theta^{\tilde{k}^*}$.

want to avoid evaluating all possible $\tilde{k}$. We cannot rely on metrics such as reconstruction error or – in the case of PCA – the fraction of variance explained by the principal components to prune our search space; these metrics inform us of how closely the latent space can represent actions in the native space but do not tell us if the representable actions yield high reward nor if the latent space will indeed increase data efficiency.

SALAS provides a more guided approach to identifying the best latent space. Our key insight is that if we require that latent spaces satisfy $\tilde{\mathcal{A}}_1 \subset \cdots \subset \tilde{\mathcal{A}}_k$, then policies representable in $\tilde{\mathcal{A}}_i$ are also representable in $\tilde{\mathcal{A}}_{i+1}$. Suppose $\tilde{\mathcal{A}}_i$ can represent a high-performing policy. We then know that $\tilde{\mathcal{A}}_{i+1}$ can yield similar or greater performance, though we expect the smaller $\tilde{\mathcal{A}}_i$ to be more data efficient. Thus, SALAS seeks the smallest $\tilde{k}$ that yields performance exceeding a predefined threshold β faster than the native agent. The choice of β defines the minimum acceptable performance in $\mathcal{M}_{\text{source}}$, characterizing how much we are willing to trade off a potential increase in data efficiency with a potential decrease in final performance.

SALAS uses the heuristic in Algorithm 1 to perform a binary search over a sequence of increasing latent space dimensionalities $\mathcal{K} = (\tilde{k}_i)_{i=1}^m$. First, SALAS trains latent-$\tilde{k}_{\lfloor m/2 \rfloor}$ agents in the source environment. If the agents reach threshold β before the native agent, SALAS eliminates latent spaces with dimensionality greater than $\tilde{k}_{\lfloor m/2 \rfloor}$ and trains latent-$\tilde{k}_{\lfloor m/4 \rfloor}$ agents next. Otherwise, SALAS eliminates latent spaces with dimensionality less than or equal to $\tilde{k}_{\lfloor m/2 \rfloor}$, and trains latent-$\tilde{k}_{\lfloor 3m/4 \rfloor}$ agents next. The binary search continues until the smallest latent space that achieves performance β faster than the native agent is found, or if no such latent space exists, the heuristic suggests that PCA-based latent action spaces are unhelpful for the environment of interest. This heuristic reduces the number of latent spaces to evaluate in $\mathcal{M}_{\text{source}}$ from $|\mathcal{K}|$ to $\log_2 |\mathcal{K}|$.

Algorithm 1: SALAS's binary search heuristic to determine the most data-efficient latent action space

```
Inputs: latent dimensionalities
  $\mathcal{K} = (\tilde{k}_i)_{i=1}^m$, performance threshold $\beta$,
  and timesteps required for the native
  agent to reach performance threshold $t_\beta$
$lo \leftarrow 1$
$hi \leftarrow m$
while $lo < hi$ do
    $mid \leftarrow \lfloor \frac{lo+hi}{2} \rfloor$
    $\tilde{k} \leftarrow \tilde{k}_{mid}$
    Train latent-$\tilde{k}$ agent in $\mathcal{M}_{\text{source}}$
    $t_\beta^{\tilde{k}} \leftarrow$ timesteps required for the agent
      to reach performance $\beta$
    if $t_\beta^{\tilde{k}} < t_\beta$ then
        $hi \leftarrow mid$
    else
        $lo \leftarrow mid + 1$
    end
end
Return: $\tilde{k}_{hi}$
```

4.2 Learning Latent Action Spaces

SALAS requires a dimensionality reduction technique to discover latent action spaces from source environment trajectories. As described in Section 4.1, SALAS requires that the latent action spaces $\tilde{\mathcal{A}}_1, \ldots, \tilde{\mathcal{A}}_k$ form a nested sequence of spaces $\tilde{\mathcal{A}}_1 \subset \tilde{\mathcal{A}}_2 \subset \cdots \subset \tilde{\mathcal{A}}_k$. We propose to use PCA (Bishop, 2006, Chapter 12), a classical dimensionality reduction technique that finds a linear projection that maximizes the variance of projected data and minimizes the projection error. While autoencoders are a popular tool for dimensionality reduction, we cannot guarantee that autoencoders trained with different latent dimensionalities will form a nested sequence of spaces. In what follows, we describe how we use PCA to learn a nested sequence of latent action spaces.

First, we train a native agent to convergence in $\mathcal{M}_{\text{source}}$ and then collect trajectories of the trained agent in $\mathcal{M}_{\text{source}}$. We combine all actions in the sampled trajectories in an $n \times k$ dataset $\mathcal{D}$ of n native actions. Given $\mathcal{D}$ and a desired latent space dimensionality $\tilde{k}$, PCA outputs an orthonormal $k \times \tilde{k}$ matrix $\boldsymbol{W}_{\tilde{k}}$ whose columns are the first $\tilde{k}$ principal components of $\mathcal{D}$ and a vector $\boldsymbol{\mu}$ equal to the mean of $\mathcal{D}$. Without loss of generality, we assume that the i^{th} column of $\boldsymbol{W}_{\tilde{k}}$ is the i^{th} principal component. The span of $\boldsymbol{W}_{\tilde{k}} + \boldsymbol{\mu}$ is the latent space $\tilde{\mathcal{A}}_{\tilde{k}}$, and the latent-to-native mapping is $l(\tilde{\boldsymbol{a}}) = \boldsymbol{W}_{\tilde{k}}\tilde{\boldsymbol{a}} + \boldsymbol{\mu}$. By construction, PCA produces a nested sequence of latent spaces $\tilde{\mathcal{A}}_1 \subset \tilde{\mathcal{A}}_2 \subset \cdots \subset \tilde{\mathcal{A}}_k$.

We have found that one seemingly minor implementation detail requires careful consideration when using PCA to learn the latent-to-native mapping l. Implementations of RL algorithms such as DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018) often bound policy output to the unit hypercube $\tilde{\mathcal{A}} \subseteq [-1,1]^{\tilde{k}}$ by applying a squashing function such as $\tanh$ to the policy output. If the action space is bounded to the hypercube, the PCA-based latent-to-native mapping will map latent actions $\tilde{\boldsymbol{a}} \in \tilde{\mathcal{A}}_{\tilde{k}}$ with $||\tilde{\boldsymbol{a}} + \boldsymbol{\mu}||_2 > 1$ to actions outside of $[-1,1]^k$, and since l is one-to-one, the action space is reduced more than intended. Figure 3 illustrates this effect in a squashed two-dimensional space. To prevent unintentional restriction of the latent space, SALAS performs PCA on a dataset of *unsquashed* actions by applying $\operatorname{arctanh}$ to actions in $\mathcal{D}$ to remove any squashing before running PCA. During latent policy optimization, we apply the squashing function to the output of l to obtain the final action to send to the environment.

5 Experiments

In this section, we present an empirical analysis of SALAS. We design our empirical analysis to answer two questions:

1. Can a latent action space learned from a single simulated environment increase data efficiency when learning over a distribution of similar environments with different physics parameters?
2. Given a set of latent action spaces, can SALAS determine which spaces will increase data efficiency without exhaustively training on all of them?

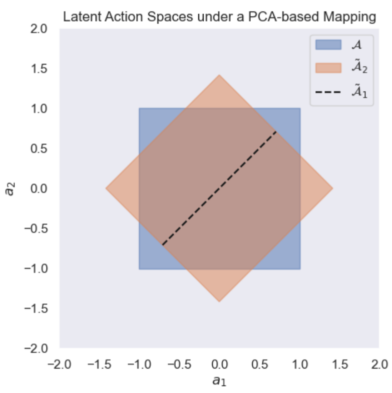


Figure 3: An illustration of how the latent-to-native mapping $l(\tilde{\boldsymbol{a}}) = \boldsymbol{W}_{\tilde{k}}\tilde{\boldsymbol{a}} + \boldsymbol{\mu}$ cannot represent some native actions with a squashed action space $\mathcal{A} = [-1, +1]^2$. In this example, $\boldsymbol{W}_2$ is a 45 degree rotation matrix, and $\boldsymbol{\mu} = (0, 0)^\top$. Without loss of generality, we take the first column of $\boldsymbol{W}_2$ to be $\boldsymbol{W}_1 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^\top$. For both $\tilde{k} = 1$ and 2, actions near the corners of the action space cannot be reached under l.

5.1 Environment Descriptions

We perform experiments using modified implementations of the OpenAI Gym MuJoCo environments (Brockman et al., 2016). **ReacherTracker20-v3** is an extension of the Reacher-v2 environment with 20 links and 20 degrees of freedom. The goal is to track an elliptical trajectory using its end-effector, similar to the robotic arm environment used by Luck et al. (2014). This environment exhibits a high level of redundancy in its control space, and we expect the actions dimensions of an optimal policy to be highly correlated. **Swimmer20-v3** is an extension of the Swimmer-v3 environment with 20-links and 19 degrees of freedom. We expect actions dimensions of an optimal policy to be correlated in this environment as well, though it requires more complex movements than ReacherTracker20-v3 to achieve good performance. We additionally use the 8-degree-of-freedom **Ant-v3** environment without modification. See Appendix A for additional environment details.

To create target environment distributions, we randomize simulation parameters at the beginning of each episode. Table 1 lists the randomized parameters for each environment as well as their randomization bounds and their default values in $\mathcal{M}_{\text{source}}$. New parameter values are sampled uniformly at random from their respective randomization bounds. To determine reasonable randomization bounds, we simulated rollouts of a trained native agent while randomizing each environment parameter individually, and we chose the tightest bounds to which the native agent could not immediately generalize.

Table 1: Randomization bounds for environment parameters. The density, damping, and slope bounds represent minimum and maximum possible values, whereas the bounds for mass, friction, and gravity represent the minimum and maximum scaling of each parameter. Density refers to the density of the environment medium used to simulate drag forces. In ReacherTracker20, the masses of each link are scaled equally. In Ant-v3, only the mass of the ant's torso is randomized.

Environment	Parameters	Randomization Bounds	Default Value
ReacherTracker20-v3	mass	[0.5, 2]	1
	damping	[1, 10]	1
Ant-v3	torso mass	[0.5, 2]	1
	friction	[0.5, 1.5]	1
	slope along x axis	[-0.1, 0.1]	0
	gravity	[0.5, 2]	1
Swimmer20-v3	density	[2000, 6000]	4000
	damping	[1, 5]	1

We use TD3 (Fujimoto et al., 2018) as the RL algorithm. We first train 10 native agents in $\mathcal{M}_{\text{source}}$ and then sample 1000 trajectories from the top performing native agent in $\mathcal{M}_{\text{source}}$ to create a dataset $\mathcal{D}$ of actions. To discover latent action spaces, we then perform PCA on unsquashed actions in $\mathcal{D}$ as described in Section 4.2. For the ReacherTracker20-v3 and Ant-v3 environments, agents are trained over 2 million timesteps, and for Swimmer20-v3, agents are trained for 1 million timesteps. We evaluate agents over 100 episodes every 10,000 timesteps during training. When training agents in the source and target environment distribution, we train over 10 and 20 random seeds, respectively. To account for differences in initialization between native and latent agents due to the additive $\boldsymbol{\mu}$, we additionally train native agents with $l(\tilde{\boldsymbol{a}}) = \tilde{\boldsymbol{a}} + \boldsymbol{\mu}$.

5.2 Results

Before evaluating if SALAS increases data efficiency in the target environment distribution, we first verify how much dimensionality reduction is feasible while still preserving actions of a high-performing policy in the source environment. For each environment, we run PCA and calculate how much variance in $\operatorname{arctanh} \mathcal{D}$ can be explained by each latent space. Figure 4 shows the explained variance for each choice of $\tilde{k}$. In all environments, at least 90% of the variance can be explained using a latent dimesionality of 4. This result suggests that high-performing policies in all environments can be approximately represented with lower-dimensional action spaces.

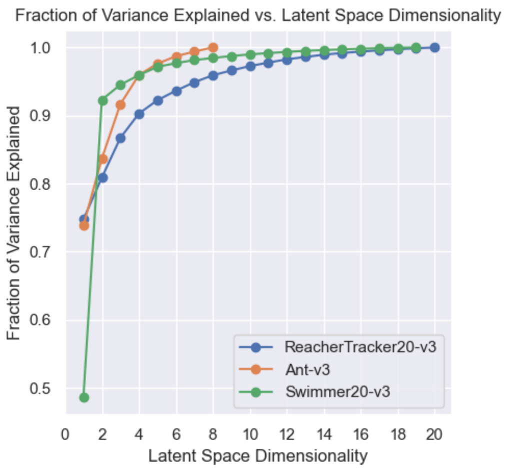


Figure 4: Fraction of variance explained for PCA-based latent spaces of increasing dimensionality.

The ability to represent the actions of the optimal policy will not necessarily translate into faster learning, however. To predict which latent action space will yield the best data efficiency in the target environment distribution, we evaluate native and latent agent data efficiency in the source environment using the heuristic described in Algorithm 1. We use the performance threshold

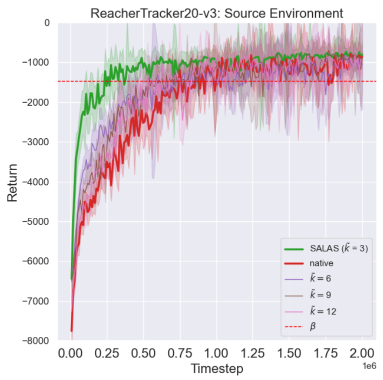
$$\beta = r_{\text{random}} + 0.9(r_{\text{native}} - r_{\text{random}}),$$

where r_{random} and r_{native} are the median return achieved by a random policy and a trained native policy in $\mathcal{M}_{\text{source}}$, respectively. Figure 5 shows training curves for different latent spaces within each source environment. We show training curves only for latent spaces that SALAS's heuristic evaluates, though training curves for all latent agents can be found in Appendix B. Due to the high variance of training, we plot the median performance in all figures. Plots showing average performance curves can also be found in Appendix B.

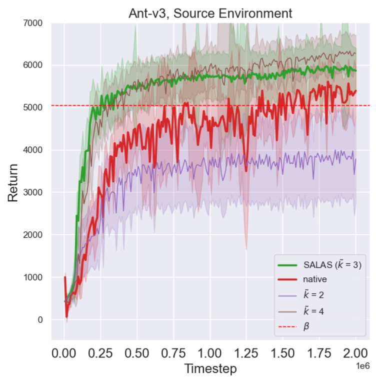
For ReacherTracker20-v3 and Ant-v3, we consider $\mathcal{K} = (3, 6, 9, 12, 15, 18, 20)$ and $\mathcal{K} = (1, 2 \ldots, 8)$, respectively. In both environments, the heuristic finds that dimensionality $\tilde{k} = 3$ yields the most data efficient latent space in $\mathcal{M}_{\text{source}}$. For Swimmer20-v3, we consider $\mathcal{K} = \{10, 12, 14, 16, 18\}$. None of the latent agents evaluated by SALAS's heuristic reach the threshold faster than the native agents, and SALAS predicts that none of the PCA-based latent action spaces will transfer well to $p_{\mathcal{M}}$. Even though latent agents in Swimmer20-v3 converge to equivalent performance as the native agent, they require more samples to do so. This observation underscores that the most data efficient latent action space is not necessarily the smallest space that can simply represent a high-performing policy.

We now answer our two primary empirical questions by training latent agents in $p_{\mathcal{M}}$ and comparing their data efficiency and final learning performance. We additionally take the native and SALAS agents trained in $\mathcal{M}_{\text{source}}$ and fine-tune them in $p_{\mathcal{M}}$ as baselines methods. We fine-tune the 10 source-trained native and the 10 source-trained SALAS agents over 5 random seeds. Performance curves are shown in Figure 6. Again, we only show training curves for latent spaces SALAS's heuristic evaluates, though curves for all latent agents can be found in Appendix B.

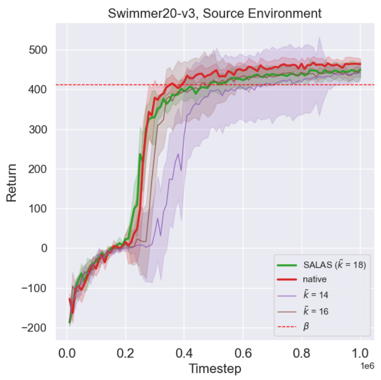
In ReacherTracker20-v3, the latent-3 agent converges after less than 500 thousand timesteps and achieves higher return than the native agent, fine-tuned agents, and all other latent agents considered. Even after 2 million timesteps, the native agent has not converged. In the source environment, the native agent could match latent-3's performance after roughly 1 million timesteps, highlighting the difficulty introduced by randomizing environment parameters. We observe a smooth increase in data efficiency as the latent space dimensionality decreases. While the fine-tuned SALAS agent maintained consistent performance throughout training, some of the fine-tuned native agents experienced a significant drop in performance near the end of training, ultimately achieving performance on par with a random policy. In Ant-v3, all latent agents except for $\tilde{k} = 1, 8$ outperform the native agent. The latent-3 agent outperforms all latent agents considered and achieves roughly double the median return achieved by the native agent. Both fine-tuned



(a) ReacherTracker20-v3

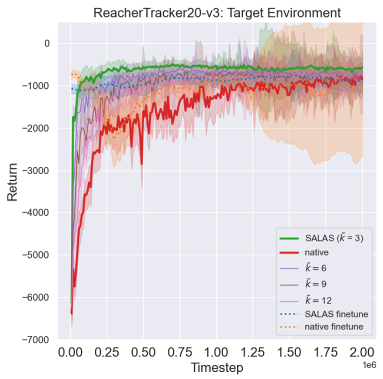


(b) Ant-v3

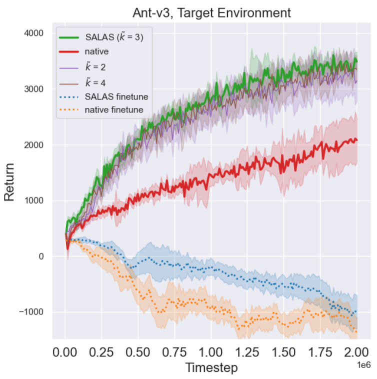


(c) Swimmer20-v3

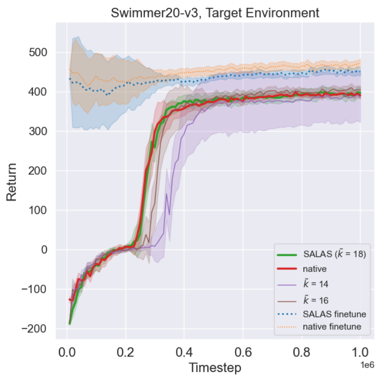
Figure 5: Training curves for agents in the source environment. Agents are evaluated over 100 episodes every 10,000 timesteps. The curves are the median return over 10 random seeds. The shaded region denotes a 90% confidence belt. The red dashed line in each plot indicates the performance threshold β.



(a) ReacherTracker20-v3



(b) Ant-v3

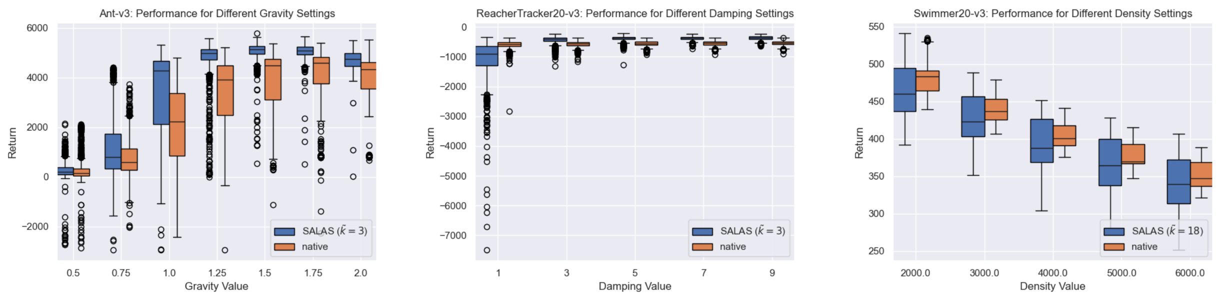


(c) Swimmer20-v3

Figure 6: Training curves for agents in the target environment distribution. Agents are evaluated over 100 episodes every 10,000 timesteps. The curves are the median over 20 random seeds. Fine-tuned baselines show 50 seeds. The shaded region denotes a 90% confidence belt.

baselines achieve poor performance. In Swimmer20-v3, latent-18 outperforms the other latent agents but does not improve data efficiency compared the native agent, matching the heuristic's prediction that none of the PCA-based latent spaces considered will offer improvement. We observe that some fine-tuned agents experience a significant drop in performance early in training though ultimately outperform all other agents.

Our results support the heuristic's utility: SALAS correctly identifies which latent space offers the greatest increase in data efficiency when learning in the target distribution, or indicates that none of them increase data efficiency, without exhaustively training with all of them. The results for ReacherTracker20-v3 and Ant-v3 further demonstrate that a latent space learned in a fixed simulator instance can greatly improve data efficiency in a distribution of similar environments. Thus, we answer the two questions posed at the beginning of this section in the affirmative. We further conclude that SALAS is preferable to fine-tuning, since fine-tuning offers inconsistent performance and sometimes fails to learn at all.



(a) Ant-v3: Friction (b) ReacherTracker20-v3: Damping (c) Swimmer20-v3: Density

Figure 7: Box plots comparing native and SALAS agent performance in each environment for various simulator parameter settings. For each parameter setting, we obtain data for each box plot by evaluating all 20 native and SALAS latent agents over 50 episodes each. In each figure, all simulation parameters are set to their default value except the parameter being plotted.

5.3 Evaluating Latent Policies

Naturally, some environments within the target environment distribution may be more difficult to learn than others. Since our RL agents maximize expected cumulative reward, agents may learn to perform well over only a portion of the target environment distribution corresponding to the easier instances and perform poorly in the more difficult instances. Moreover, it is possible that latent agents would struggle to learn on environments that differ substantially from the source environment.

To assess how well SALAS's action space generalizes outside of the source environment, we perform a finer-grained evaluation of latent policies by observing how performance changes as simulator parameters deviate further and further from the source environment. For a given simulator parameter, we choose several evenly-spaced values within its randomization bounds, and then instantiate (fixed) environment instances with these parameter values. All other parameters are kept at their default values. We then evaluate native and latent agents trained in $p_{\mathcal{M}}$ on these instances. We only evaluate the latent agents chosen by SALAS, since they provide the largest increase in data efficiency. If SALAS is overly-specialized to environments similar to the source environment, we expect that latent agent performance will drop as environment parameters deviate more significantly from the source environment parameters.

Figure 7 shows how latent agent performance changes as one physics parameter is changed in each environment. See Appendix C for additional figures showing how performance changes when the remaining physics parameters are varied. In Ant-v3, both native and SALAS agents achieve low return for smaller gravity values and higher return for larger gravity values. Since we observe this trend for both the native and SALAS agents, this specialization is likely due to the inherent difficult of the low-gravity environment instances rather than an overly-specialized latent action space. Interestingly, in ReacherTracker20-v3, the SALAS agents outperform the native agent only when the damping coefficient is larger than 1, even though a damping coefficient of 1 is used in the source environment. As shown in Figure 5, the SALAS agent can indeed match native performance in the source environment. Since the SALAS agents achieve higher median performance than the native agent in $p_{\mathcal{M}}$, we hypothesize that this seeming specialization is caused by the SALAS agents exploiting the easier environment instances with larger damping values.

6 Conclusions and Future Work

We introduced SALAS, a technique for improving data efficiency when training over a distribution of environments. SALAS learns a set of lower-dimensional latent action spaces in a single simulator instance, uses a heuristic to predict which latent space will yield the greatest increase in data efficiency in the target distribution, and then trains over the target distribution using the most promising latent space. We evaluated SALAS over several MuJoCo environments (Brockman et al., 2016) with randomized physics parameters. Empirical results indicated that SALAS can substantially improve data efficiency compared to training in the original action space. Moreover, we observed that SALAS's heuristic correctly chose the latent spaces that yield the greatest data efficiency and correctly predicted when none of the latent actions spaces considered improve data efficiency. Overall, SALAS can be used to efficiently train agents to perform well across a variety of environmental conditions.

Interestingly, we find that latent action spaces may reduce data efficiency even if they can represent policies that perform on par with agents trained in the original action space. While SALAS can identify such latent spaces prior to training in the target distribution, it remains unclear why data efficiency decreases. These findings raise questions about whether dimensionality reduction necessarily results in an easier learning problem. Recent work by Dabney et al. (2021) shows that effective value function representations are not only able to represent the optimal value function but also the optimal sequence of value functions leading to it. It would be interesting to investigate if a similar result applies for action space representations. Furthermore, extending our approach to non-linear dimensionality reduction techniques (*e.g.* using autoencoders) is of interest, since SALAS relies on the properties possessed by linear PCA-based latent action spaces. Though we found linear dimensionality reduction to be sufficient, low-dimensional structure may be better representable using non-linear methods.

7 Acknowledgements

Support for this research was provided by the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin — Madison with funding from the Wisconsin Alumni Research Foundation. We would like to thank John Balis and Asmit Nayak for helpful feedback on earlier drafts.

References

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, pp. 1–8, 2019.

Arthur Allshire, Roberto Mart'in-Mart'in, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. Laser: Learning a latent action space for efficient reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6650–6656, 2021.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006. ISBN 0387310738.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 941–950, 2019.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D. Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, 2019.

Paul Francis Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, P. Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *ArXiv*, abs/1610.03518, 2016.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521:503–507, 2015.

Mark Cutler and Jonathan P. How. Efficient reinforcement learning for robots using informative simulated priors. In *International Conference on Robotics and Automation (ICRA)*, pp. 2605–2612, 2015.

Mark Cutler, Thomas J Walsh, and Jonathan P How. Reinforcement learning with multi-fidelity simulators. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3888–3895, 2014.

Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1331–1340, 2019.

Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 35, 2021.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pp. 1582–1591, 2018.

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018.

Josiah P Hanna, Siddharth Desai, Haresh Karnan, Garrett Warnell, and Peter Stone. Grounded action transformation for sim-to-real reinforcement learning. *Machine Learning*, 110(9):2469–2499, 2021.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations (ICLR)*, 2018.

Ayush Jain, Andrew Szot, and Joseph Lim. Generalization to new actions in reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4661–4672. PMLR, 13–18 Jul 2020.

Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning (CoRL)*, pp. 334–343, 2017.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Svetoslav Kolev and Emanuel Todorov. Physically consistent state estimation and system identification for contacts. In *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 1036–1043, 2015.

Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *GECCO*, 2010.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Dylan P. Losey, Krishna Parasuram Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–384, 2020.

Kevin Sebastian Luck, Gerhard Neumann, Erik Berger, Jan Peters, and Heni Ben Amor. Latent space policy search for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1434–1440, 2014.

Kevin Sebastian Luck, Joni Pajarinen, Erik Berger, Ville Kyrki, and Heni Ben Amor. Sparse latent space policy search. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1911–1918, 2016.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher Joseph Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning (CoRL)*, 2019.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, 2018.

Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma (eds.), *Robotics: Science and Systems XIII*, 2017.

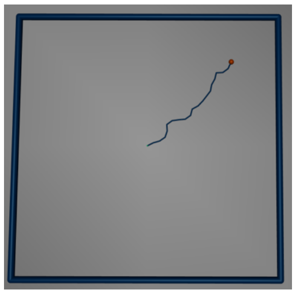
John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

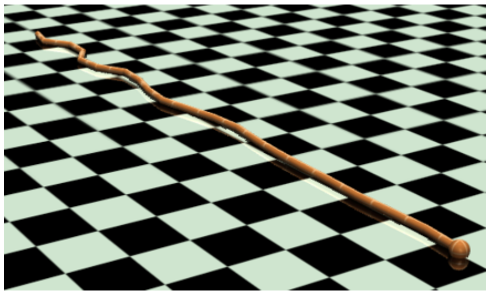
Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning (ICML)*, pp. 9767–9779. PMLR, 2021.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Guy Tennenholtz and Shie Mannor. The natural language of actions. In *International Conference on Machine Learning (ICML)*, pp. 6196–6205, 2019.

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. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.


(a) ReacherTracker20-v3


(b) Swimmer20-v3

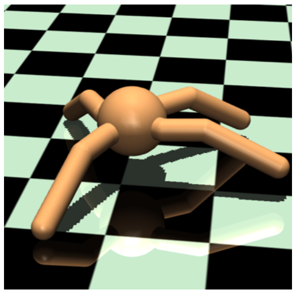

(c) Ant-v3

Figure 8: Simulation renderings of the agent in each environment.

A ENVIRONMENT DESCRIPTIONS

We provide more detail on each environment considered in our experiments.

ReacherTracker20-v3: The ReacherTracker20-v3 environment (Figure 8a) is a 20-link extension of the Reacher-v2 environment with 20 degrees of freedom and state dimension of 29. The goal is to track the elliptical path

$$\boldsymbol{g} = \begin{pmatrix} 0.2\cos(2\pi t/200) + 1.4 \\ 0.4\sin(2\pi t/200) + 0.8 \end{pmatrix}, \quad t = 1, \dots 200$$

over 200 timesteps using the reacher's end-effector. Each link is 0.1 units long. At the beginning of each episode, each joint angle is initialized to a value in $[-0.1, +0.1]$ uniformly at random. The reward function is

$$r(\boldsymbol{s}, \boldsymbol{a}) = -c||\boldsymbol{a} - \boldsymbol{g}|| - ||\boldsymbol{a}||^2$$

where $c = 20$. The original Reacher-v2 environment uses $c = 1$, though we found it necessary to use $c = 20$, otherwise the agent would purely maximize $-||\boldsymbol{a}||^2$ without solving the tracking task.

Swimmer20-v3: The Swimmer20-v3 environment (Figure 8b) is a 20-link extension of the Swimmer-v2 environment with 19 degrees of freedom and state dimension of 40. The goal is travel as fast as possible in along the horizontal axis. The the agent's head is restricted to slide only along the horizontal axis. At the beginning of each episode, each joint angle is initialized to a value in $[-0.1, +0.1]$. The environment is otherwise identical to the original Swimmer-v2 environment.

Ant-v3: In the Ant-v3 environment (Figure 8c), an 8 degree-of-freedom quadruped must learn to walk along the horizontal axis. The state space has 111 dimensions. Aside from modifying physics parameters, we make no modifications to the original Ant-v3 environment.

B Additional Experiments

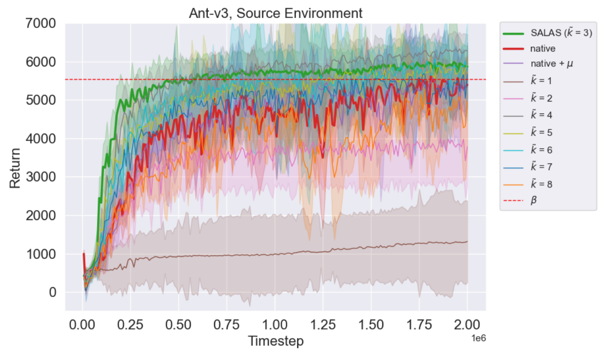
In this section, we show performance curves for latent and native agents not shown in the main text. Figure 9 shows the median performance for all native and latent agents considered in our experiments. We additionally provide the average performance curves in Figure 10. Average final performance in $\mathcal{M}_{\text{source}}$ and $p_{\mathcal{M}}$ are shown in Figures 11 and 12, respectively.

When plotting average returns in the source environment, we use the following modified performance threshold:

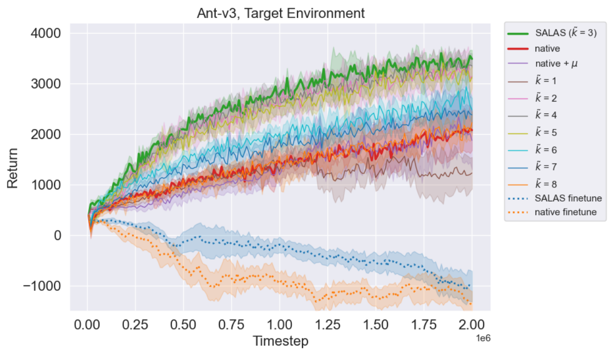
$$\beta = \bar{r}_{\text{random}} + 0.9(\bar{r}_{\text{native}} - \bar{r}_{\text{random}}),$$

where $\bar{r}_{\text{random}}$ and $\bar{r}_{\text{native}}$ are the *average* return achieved by a random policy and a trained native policy in $\mathcal{M}_{\text{source}}$, respectively. In both Figures 9 and 10, we plot 10 and 20 random seeds for agents trained in the source environment and target environment distribution, respectively. The shaded region denotes a 90% confidence belt. Each agent is evaluated over 100 episodes every 10,000 timesteps.

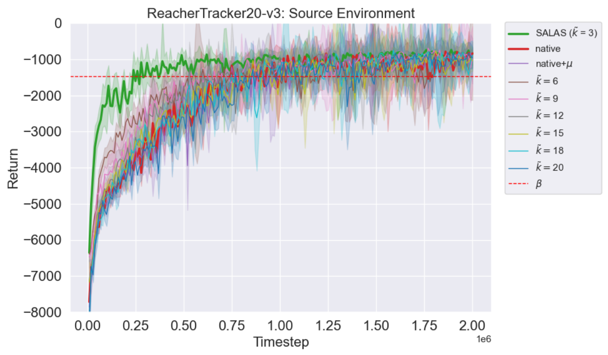
In Ant-v3, we observe a increase in data efficiency as $\tilde{k}$ increases to 3 followed by a decrease as $\tilde{k}$ increases further. In ReacherTracker20-v3, data efficiency increases as $\tilde{k}$ decreases to 3, and in Swimmer20-v3, performance decreases as $\tilde{k}$ increases from 18. These observations verify that SALAS indeed selects the most data efficient $\tilde{k}$ from the set of considered latent dimensionalities.



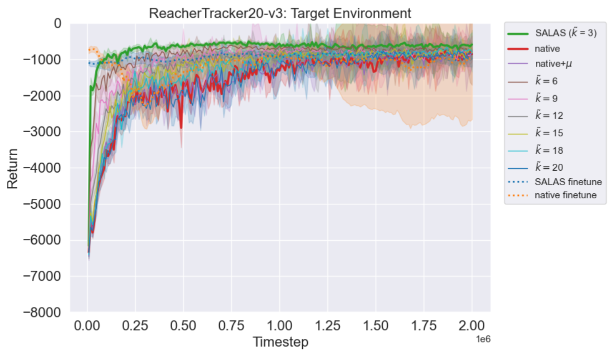
(a) Ant-v3: Source Environment



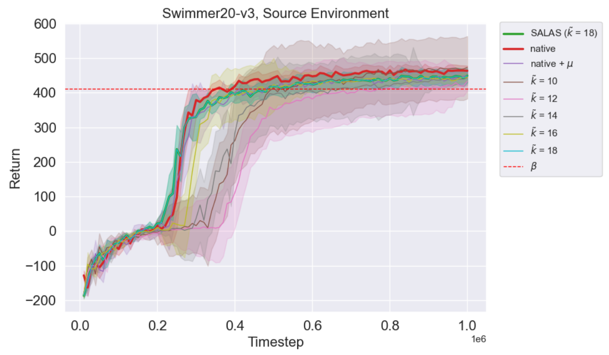
(b) Ant-v3: Target Environment Distribution



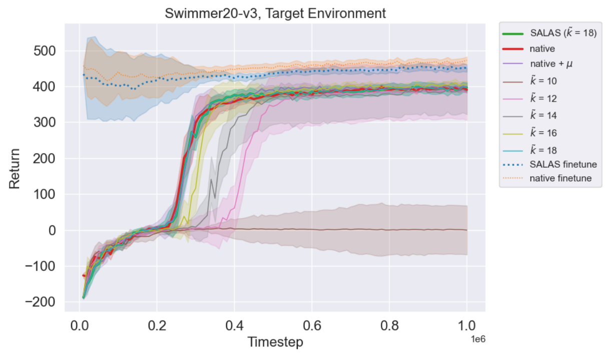
(c) ReacherTracker20-v3: Source Environment



(d) ReacherTracker20-v3: Target Environment Distribution

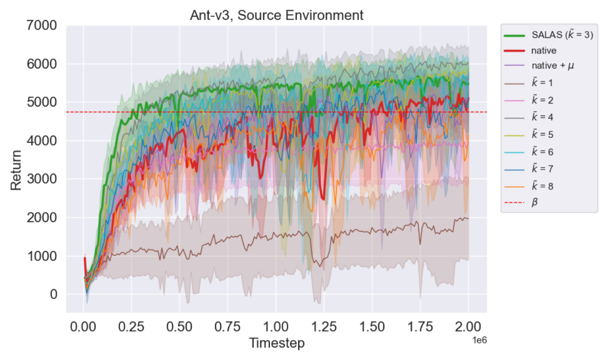


(e) Swimmer20-v3: Source Environment

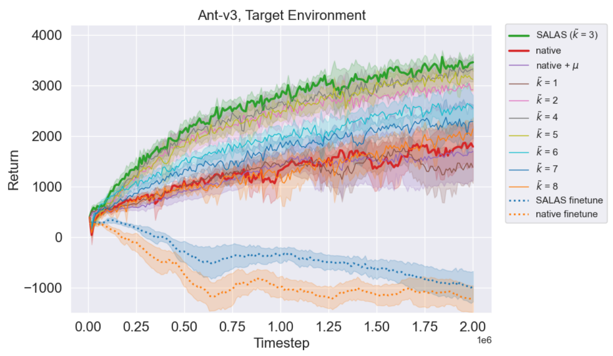


(f) Swimmer20-v3: Target Environment Distribution

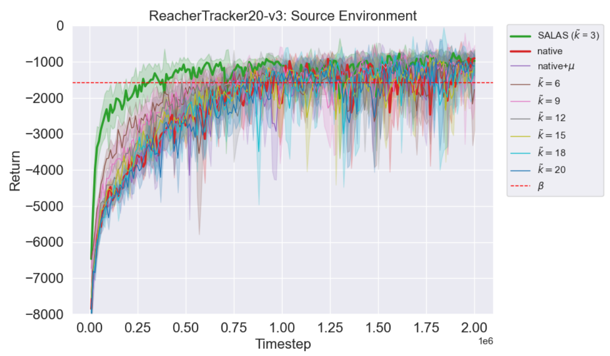
Figure 9: Median performance curves for latent and native agents in $\mathcal{M}_{\text{source}}$ and $p_{\mathcal{M}}$.



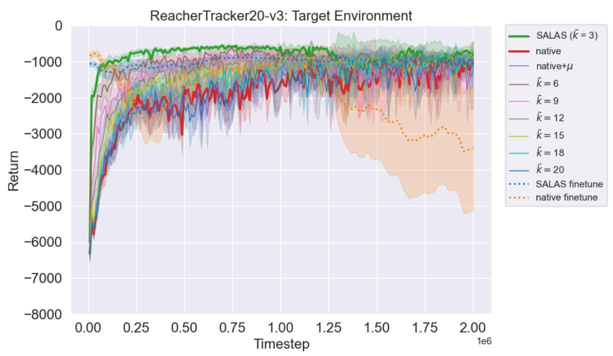
(a) Ant-v3: Source Environment



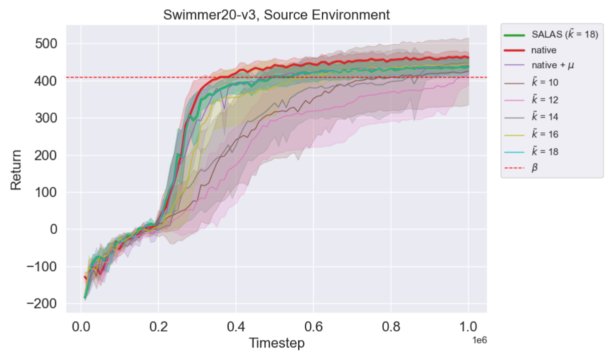
(b) Ant-v3: Target Environment Distribution



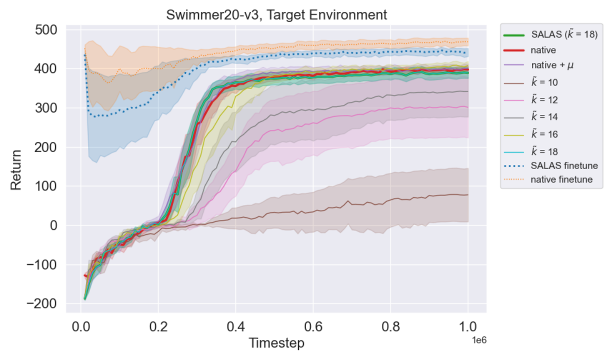
(c) ReacherTracker20-v3: Source Environment



(d) ReacherTracker20-v3: Target Environment Distribution

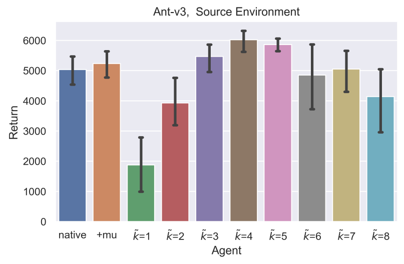


(e) Swimmer20-v3: Source Environment

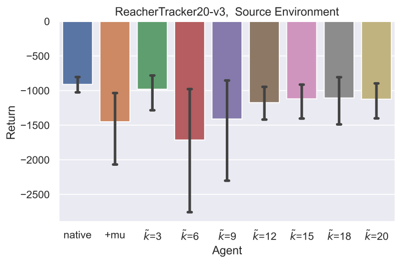


(f) Swimmer20-v3: Target Environment Distribution

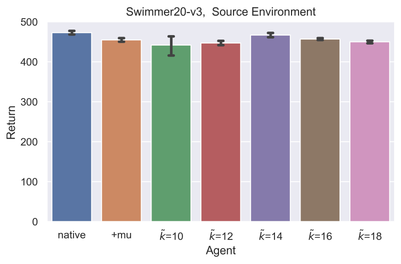
Figure 10: Average performance curves for latent and native agents in $\mathcal{M}_{\text{source}}$ and $p_{\mathcal{M}}$.



(a) ReacherTracker20-v3

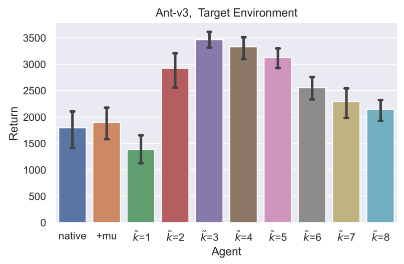


(b) Ant-v3

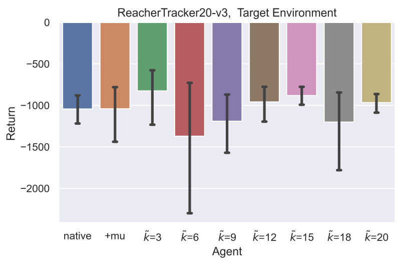


(c) Swimmer20-v3

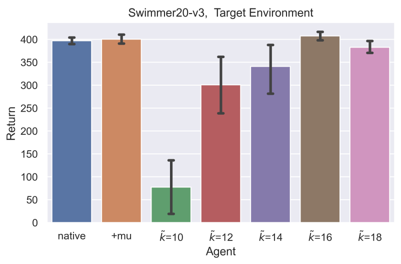
Figure 11: Average performance in $\mathcal{M}_{\text{source}}$ with error bars corresponding to 90% confidence intervals. The '+mu' abbreviates 'native + μ'.



(a) ReacherTracker20-v3



(b) Ant-v3



(c) Swimmer20-v3

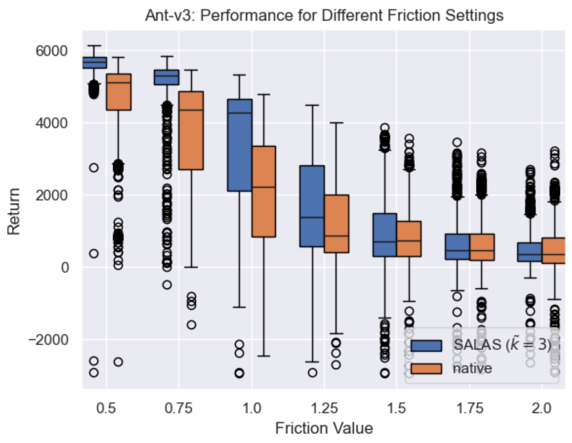
Figure 12: Average performance in $p_{\mathcal{M}}$ with error bars corresponding to 90% confidence intervals. The '+mu' abbreviates 'native + μ'.

C Evaluating Latent Policies

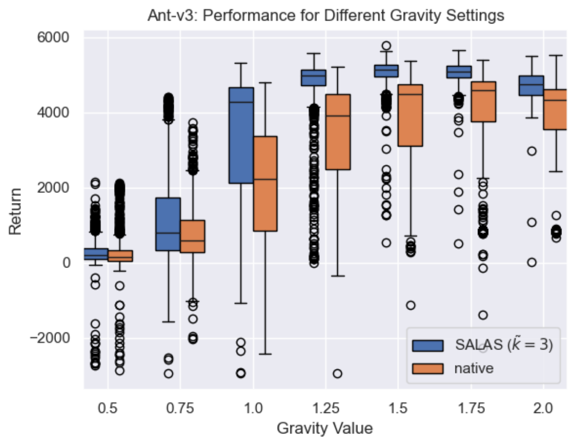
We provide additional empirical results for experiments described in Section 5.3 which we describe again here. For a single environment parameter, we choose evenly-spaced values within its randomization bounds, and then instantiate environments with these parameter values, keeping all other parameters at their default values (see Table 1). We then evaluate native and latent agents trained in $p_{\mathcal{M}}$ on these instances. We only evaluate the latent agents chosen by SALAS. Figures 13, 14, and 15 show the performance of SALAS latent agents for each environment as we vary each simulator parameter individually. We obtain data for each box plot by evaluating all 20 native and SALAS latent agents over 50 episodes each.

In Ant-v3, both native and latent agents perform poorly for larger friction values and smaller gravity values. Performance degrades more smoothly as torso mass decreases and as the slope deviates from 0. In Swimmer20-v3, native and latent agent performance remains consistent as damping varies and decreases as density increases. Native and latent agent performance follow the same trends across all parameters for Ant-v3 and Swimmer20-v3, indicating that agents are specializing to easier parameter settings in the target environment distribution.

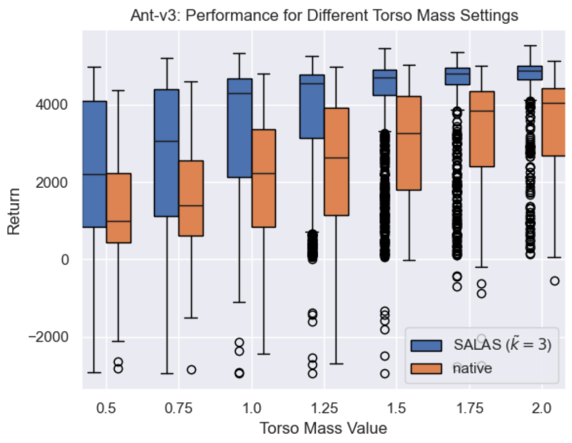
In ReacherTracker20-v3, we observe that the native agents slightly outperform the latent agents when mass is varied, though performance is consistent. Latent agents outperform native agents with a median performance of approximately -500 (Figure 9), though they only perform this well when damping is larger than 1. Thus, the latent agents are specializing to easier environment instances with easier parameter settings to achieve higher return.



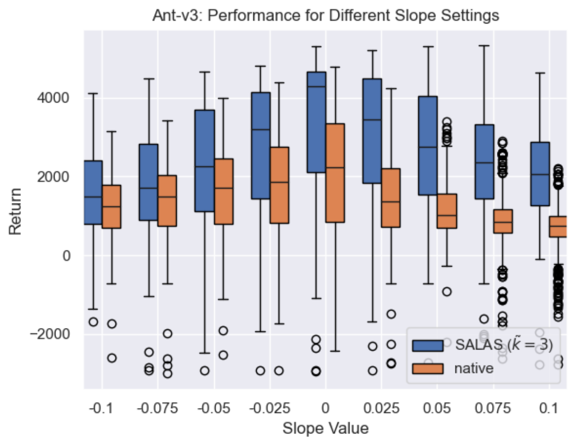
(a) Ant-v3: Friction



(b) Ant-v3: Gravity

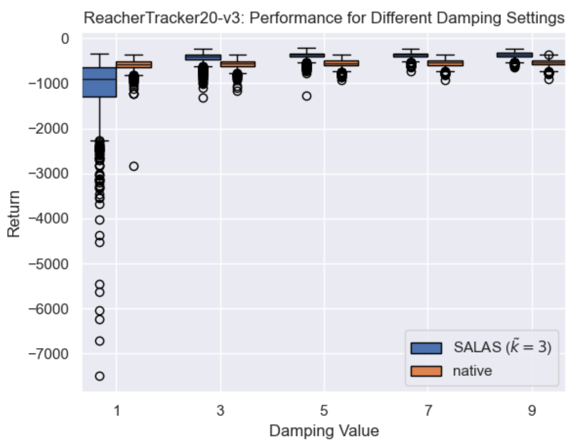


(c) Ant-v3: Torso Mass

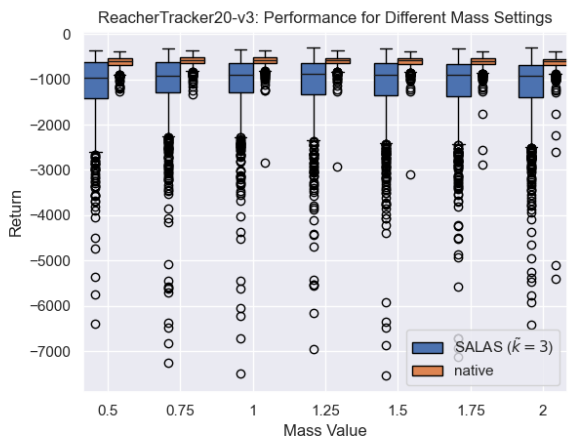


(d) Ant-v3: Slope

Figure 13: Box plots comparing native and SALAS agent performance in the Ant-v3 environment for various simulator parameter settings.

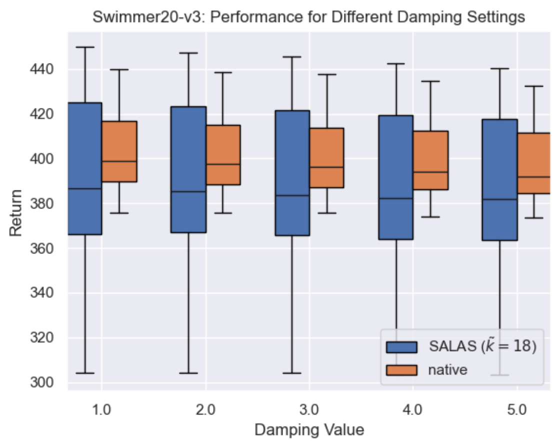


(a) ReacherTracker20-v3: Damping

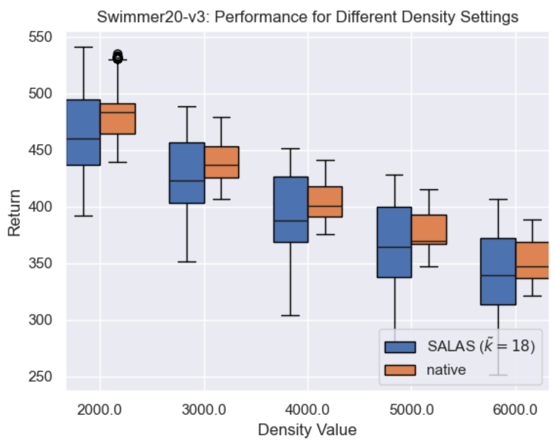


(b) ReacherTracker20-v3: Mass

Figure 14: Box plots comparing native and SALAS agent performance in the ReacherTracker20-v3 environment for various simulator parameter settings.

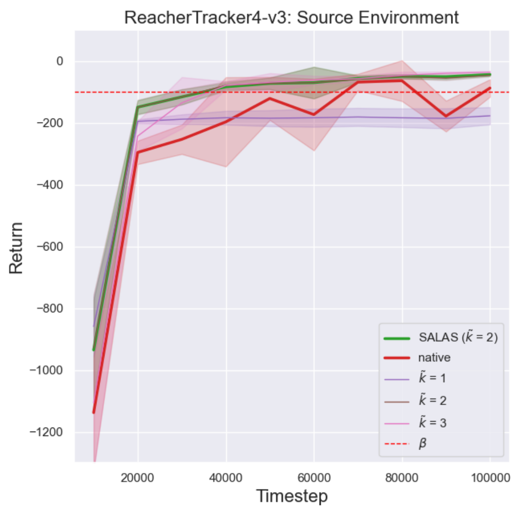


(a) Swimmer20-v3: Damping

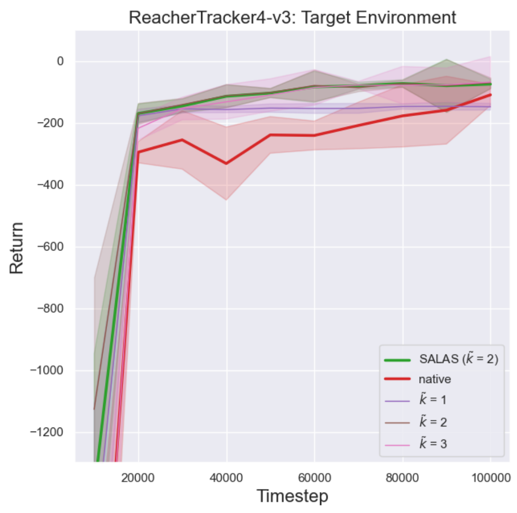


(b) Swimmer20-v3: Density

Figure 15: Box plots comparing native and SALAS agent performance in the Swimmer20-v3 environment for various simulator parameter settings.



(a) Source Environment



(b) Target Environment

Figure 16: ReacherTracker4-v3

D Reacher Environment Ablations

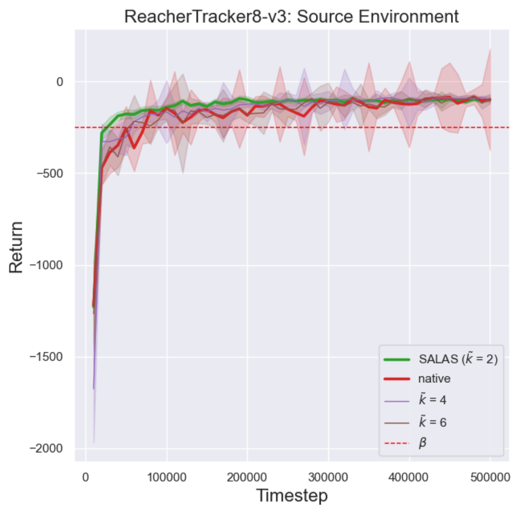
The tracking task in the ReacherTracker20-v3 environment can be solved with much fewer than 20 degrees of freedom. Nevertheless, a performance gap between native and latent agents still exists when we consider reacher environments with fewer degrees of freedom. We introduce a general ReacherTracker$k$-v3 environment, where k denotes the number of links in the reacher (and the size of the action space). The goal is to track the elliptical trajectory

$$\boldsymbol{g} = \begin{pmatrix} 0.01k\cos(2\pi t/h) + 0.07k \\ 0.02k\sin(2\pi t/h) + 0.14k \end{pmatrix}, \quad t = 1, \dots h$$

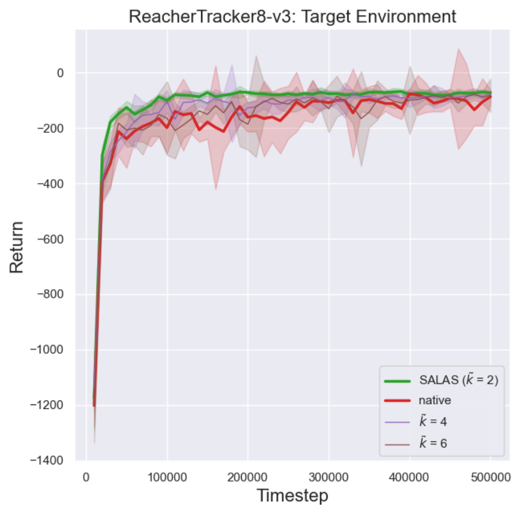
over h timesteps using the reacher's end-effector. We apply SALAS to ReacherTracker8-v3 and ReacherTracker4-v3. For $k = 8$, we use $h = 200$ as in the original ReacherTracker20-v3 environment. For $k = 4$, we use $h = 600$ to slightly increase task difficulty, otherwise all agents would solve the task in very few timesteps.

We use the same source and target environment distribution physics parameters used for ReacherTracker20-v3 (see Table 1). During training, we evaluate agents over 100 episodes every 10,000 timesteps. In the source and target environment distribution, we train over 10 and 20 random seeds, respectively. Native agents are trained for 1 million timesteps in ReacherTracker8-v3 and for 200 thousand timesteps in ReacherTracker4-v3, and we use the policy with the best overall performance to create the native action set $\mathcal{D}$. We train latent agents for half as many timesteps.

Results are shown in 17 and 16. The curves denote the median over 10 random seeds. The shaded region denotes a 90% confidence belt. In both environments, SALAS chooses $\tilde{k} = 2$. We observe that a latent-2 agent converges the fastest and exhibits less variance than a native agent in both $\mathcal{M}_{\text{source}}$ and $p_{\mathcal{M}}$.



(a) Source Environment



(b) Target Environment

Figure 17: ReacherTracker8-v3