Latent Space Policies for Hierarchical Reinforcement Learning

Tuomas Haarnoja [* 1]   **Kristian Hartikainen** [* 2]   **Pieter Abbeel** [1]   **Sergey Levine** [1]

Abstract

We address the problem of learning hierarchical deep neural network policies for reinforcement learning. In contrast to methods that explicitly restrict or cripple lower layers of a hierarchy to force them to use higher-level modulating signals, each layer in our framework is trained to directly solve the task, but acquires a range of diverse strategies via a maximum entropy reinforcement learning objective. Each layer is also augmented with latent random variables, which are sampled from a prior distribution during the training of that layer. The maximum entropy objective causes these latent variables to be incorporated into the layer's policy, and the higher level layer can directly control the behavior of the lower layer through this latent space. Furthermore, by constraining the mapping from latent variables to actions to be invertible, higher layers retain full expressivity: neither the higher layers nor the lower layers are constrained in their behavior. Our experimental evaluation demonstrates that we can improve on the performance of single-layer policies on standard benchmark tasks simply by adding additional layers, and that our method can solve more complex sparse-reward tasks by learning higher-level policies on top of high-entropy skills optimized for simple low-level objectives.

1. Introduction

Model-free deep reinforcement learning (RL) has demonstrated potential in many challenging domains, ranging from games (Mnih et al., 2013; Silver et al., 2016) to manipulation (Levine et al., 2016) and locomotion task (Schulman et al., 2015). Part of the promise of incorporating deep representations into RL is the potential for the emergence of hierarchies, which can enable reasoning and decision making at different levels of abstract. A hierarchical RL algorithm

could, in principle, efficiently discover solutions to complex problems and reuse representations between related tasks. While hierarchical structures have been observed to emerge in deep networks applied to perception tasks, such as computer vision (LeCun et al., 2015), it remains an open question how suitable hierarchical representations can be induced in a reinforcement learning setting. A central challenge with these methods is the automation of the hierarchy construction process: hand-specified hierarchies require considerable expertise and insight to design and limit the generality of the approach (Sutton et al., 1999; Kulkarni et al., 2016; Tessler et al., 2017), while automated methods must contend with severe challenges, such as the collapse of all primitives into just one useful skill (Bacon et al., 2017) or the need to hand-engineer primitive discovery objectives or intermediate goals (Heess et al., 2016).

When learning hierarchies automatically, we must answer a critical question: What objective can we use to ensure that lower layers in a hierarchy are useful to the higher layers? Prior work has proposed a number of heuristic approaches, such as hiding some parts of the observation from the lower layers (Heess et al., 2016), hand-designing state features and training the lower layer behaviors to maximize mutual information against these features (Florensa et al., 2017), or constructing diversity-seeking priors that cause lower-layer primitives to take on different roles (Daniel et al., 2012; Eysenbach et al., 2018). Oftentimes, these heuristics intentionally cripple the lower layers of the hierarchy, for example, by withholding information, so as to force a hierarchy to emerge, or else limit the higher levels of the hierarchy to selecting from among a discrete set of skills (Bacon et al., 2017). In both cases, the hierarchy is forced to emerge because neither higher nor lower layers can solve the problem alone. However, constraining the layers in this way can involve artificial and task-specific restrictions (Heess et al., 2016) or else diminish overall performance.

Instead of crippling or limiting the different levels of the hierarchy, we can imagine a hierarchical framework in which each layer directly attempts to solve the task and, if it is not fully successful, makes the job easier for the layer above it. In this paper, we explore a solution to the hierarchical reinforcement learning problem based on this principle. In our framework, each layer of the hierarchy corresponds to a policy with internal latent variables. These latent vari-

---

[*]Equal contribution [1]Berkeley Artificial Intelligence Research, University of California, Berkeley, USA [2]Independent researcher, Seattle, WA, USA. Correspondence to: Tuomas Haarnoja <haarnoja@berkeley.edu>, Kristian Hartikainen <kristian.hartikainen@gmail.com>.

ables determine how the policy maps states into actions, and the latent variables of the lower-level policy act as the action space for the higher level. Crucially, each layer is unconstrained, both in its ability to sense and affect the environment: each layer receives the full state as the observation, and each layer is, by construction, fully invertible, so that higher layers in the hierarchy can undo any transformation of the action space imposed on the layers below.

In order to train policies with latent variables, we cast the problem of reinforcement learning into the framework of probabilistic graphical models. To that end, we build on maximum entropy reinforcement learning (Todorov, 2007; Ziebart et al., 2008), where the RL objective is modified to optimize for stochastic policies that maximize both reward and entropy. It can be shown that, in this framework, the RL problem becomes equivalent to an inference problem in a particular type of probabilistic graphical model (Toussaint, 2009). By augmenting this model with latent variables, we can derive a method that simultaneously produces a policy that attempts to solve the task, and a latent space that can be used by a higher-level controller to *steer* the policy's behavior.

The particular latent variable model representation that we use is based on normalizing flows (Dinh et al., 2016) that transform samples from a spherical Gaussian prior latent variable distribution into a posterior distribution, which in the case of our policies corresponds to a distribution over actions. When this transformation is described by a general-purpose neural network, the model can represent any distribution over the observed variable when the network is large enough. By conditioning the entire generation process on the state, we obtain a policy that can represent any conditional distribution over actions. When combined with maximum entropy reinforcement learning algorithms, this leads to a RL method that is expressive, powerful, and surprisingly stable. In fact, our experimental evaluation shows that this approach can attain state-of-the-art results on a number of continuous control benchmark tasks by itself, independently of its applicability to hierarchical RL.

The contributions of our paper consist of a stable and scalable algorithm for training maximum entropy policies with latent variables as well as a framework for constructing hierarchies out of these latent variable policies. Hierarchies are constructed in layerwise fashion, by training one latent variable policy at a time, with each policy using the latent space of the policy below it as an action space, as illustrated in Figure 2. Each layer can be trained either on the true reward for the task, without any modification, or on a lower-level shaping reward. For example, for learning a complex navigation task, lower layers might receive a reward that promotes locomotion, regardless of direction, while higher layers aim to reach a particular location. When the shap-

ing terms are not available, the same reward function can be used for each layer, and we still observe significant improvements from hierarchy. Our experimental evaluation illustrates that our method produces state-of-the-art results in terms of sample complexity on a variety of benchmark tasks, including a humanoid robot with 21 actuators, even when training a single layer of the hierarchy, and can further improve performance when additional layers are added. Furthermore, we illustrate that more challenging tasks with sparse reward signals can be solved effectively by providing shaping rewards to lower layers.

2. Related Work

A number of prior works have explored how reinforcement learning can be cast in the framework of probabilistic inference (Kappen, 2005; Todorov, 2007; Ziebart et al., 2008; Toussaint, 2009; Peters et al., 2010; Neumann, 2011). Our approach is based on a formulation of reinforcement learning as inference in a graphical model (Ziebart et al., 2008; Toussaint, 2009; Levine, 2014). Prior work has shown that this framework leads to an entropy-maximizing version of reinforcement learning, where the standard objective is augmented with a term that also causes the policy to maximize entropy (Ziebart et al., 2008; Haarnoja et al., 2017; 2018; Nachum et al., 2017; Schulman et al., 2017a). Intuitively, this encourages policies that maximize reward while also being as random as possible, which can be useful for robustness, exploration, and, in our case, increasing the diversity of behaviors for lower layers in a hierarchy. Building on this graphical model interpretation of reinforcement learning also makes it natural for us to augment the policy with latent variables. While several prior works have sought to combine maximum entropy policies with learning of latent spaces (Haarnoja et al., 2017; Hausman et al., 2018) and even with learning hierarchies in small state spaces (Saxe et al., 2017), to our knowledge, our method is the first to extend this mechanism to the setting of learning hierarchical policies with deep RL in continuous domains.

Prior frameworks for hierarchical learning are often based on either options or contextual policies. The options framework (Sutton et al., 1999) combines low-level option policies with a top-level policy that invokes individual options, whereas contextual policies (Kupcsik et al., 2013; Schaul et al., 2015; Heess et al., 2016) generalize options to continuous goals. One of the open questions in both options and contextual policy frameworks is how the base policies should be acquired. In some situations, a reasonable solution is to resort to domain knowledge and design a span of subgoals manually (Heess et al., 2016; Kulkarni et al., 2016; MacAlpine & Stone, 2018). Another option is to train the entire hierarchy end-to-end (Bacon et al., 2017; Vezhnevets et al., 2017; Daniel et al., 2012). While the end-to-end training scheme provides generality and flexibility, it is prone to

learning degenerate policies that exclusively use a single option, losing much of the benefit of the hierarchical structure (Bacon et al., 2017). To that end, the option-critic (Bacon et al., 2017) adopts a standard entropy regularization scheme ubiquitous in policy gradient methods (Mnih et al., 2016; Schulman et al., 2015), Florensa et al. propose maximizing the mutual information of the top-level actions and the state distribution, and Daniel et al. bound the mutual information of the actions and top-level actions. Our method also uses entropy maximization to obtain diverse base policies, but in contrast to prior methods, our sub-policies are invertible and parameterized by continuous latent variables. The higher levels can thus undo any lower level transformation, and the lower layers can learn independently, allowing us to train the hierarchies in bottom-up layerwise fashion. Unlike prior methods, which use structurally distinct higher and lower layers, all layers in our hierarchies are structurally generic, and are trained with exactly the same procedure.

3. Preliminaries

In this section, we introduce notation and summarize standard and maximum entropy reinforcement learning.

3.1. Notation

We address policy learning in continuous action spaces formalized as learning in a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, p_{\mathbf{s}}, r)$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state and actions spaces, and $p_{\mathbf{s}} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ represents the state transition probabilities of the next state $\mathbf{s}_{t+1} \in \mathcal{S}$ given the current state $\mathbf{s}_t \in \mathcal{S}$ and action $\mathbf{a}_t \in \mathcal{A}$. At each transition, the environment emits a bounded reward $r : \mathcal{S} \times \mathcal{A} \to [r_{\min}, r_{\max}]$. We will also use $p(\mathbf{s}_0)$ to denote the initial state distribution, $\tau = (\mathbf{s}_0, \mathbf{a}_0, ..., \mathbf{s}_T)$ to denote a trajectory and $\rho_\pi(\tau)$ its distribution under a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$.

3.2. Maximum Entropy Reinforcement Learning

The standard objective used in reinforcement learning is to maximize the expected sum of rewards $\sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}[r_t]$. We will consider a more general maximum entropy objective (see, e.g. , (Todorov, 2007; Ziebart, 2010; Rawlik et al., 2012; Fox et al., 2016; Haarnoja et al., 2017; 2018)), which augments the objective with the expected entropy of the policy over $\rho_\pi(\tau)$:

$$J(\pi) = \mathbb{E}_{\tau\sim\rho_\pi(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) + \alpha\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t))\right]. \quad (1)$$

The temperature parameter $\alpha > 0$ determines the relative importance of the entropy term against the reward and thus controls the stochasticity of the optimal policy. The conventional objective can be recovered in the limit as $\alpha \to 0$. For the rest of this paper, we will omit writing the temperature

explicitly, as it can always be subsumed into the reward by scaling it with α^{-1}. In practice, we optimize a discounted, infinite horizon objective, which is more involved to write out explicitly, and we refer the interested readers to prior work for details (Haarnoja et al., 2017).

4. Control as Inference

In this section, we derive the maximum entropy objective by transforming the optimal control problem into an inference problem. Our proposed hierarchical framework will later build off of this probabilistic view of optimal control (Section 5).

4.1. Probabilistic Graphical Model for Control

Our derivation is based on the probabilistic graphical model in Figure 1(a). This model is composed of factors for the dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and for an action prior $p(\mathbf{a}_t)$, which is typically taken to be a uniform distribution but, as we will discuss later, will be convenient to set to a Gaussian distribution in the hierarchical case. Because we are interested in inferring the optimal trajectory distribution under a given reward function, we attach to each state and action a binary random variable $\mathcal{O}_t$, or *optimality variable*, denoting whether the time step was "optimal." To solve the optimal control problem, we can now infer the posterior action distribution $\pi^*(\mathbf{a}_t|\mathbf{s}_t) = p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T} = true)$, which simply states that an optimal action is such that the optimality variable is active for the current state and for all of the future states. For the remainder of this paper, we will refrain, for conciseness, from explicitly writing $\mathcal{O}_t = true$, and instead write $\mathcal{O}_t$ to denote the state-action tuple for the corresponding time was optimal.

A convenient way to incorporate reward function into this framework is to choose $p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$, assuming, without loss of generality, that $r(\mathbf{s}_t, \mathbf{a}_t) < 0$. We can write the distribution over optimal trajectories as

$$p(\tau|\mathcal{O}_{0:T}) \propto p(\mathbf{s}_0)\prod_{t=0}^{T} p(\mathbf{a}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\exp(r(\mathbf{s}_t, \mathbf{a}_t))$$

$$(2)$$

and use it to make queries, such as $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T})$. As we will discuss in the next section, using variational inference to determine $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T})$ reduces to the familiar maximum entropy reinforcement learning problem in Equation 1.

4.2. Reinforcement Learning via Variational Inference

The optimal action distribution inferred from Equation 2 cannot be directly used as a policy for two reasons. First, it would lead to an overly optimistic policy that assumes that the stochastic state transitions can also be modified to

prefer optimal behavior, even though in practice, the agent has no control over the dynamics. Second, in continuous domains, the optimal policy is intractable and has to be approximated, for example by using a Gaussian distribution. We can correct both issues by using structured variational inference, where we approximate the posterior with a probabilistic model that constrains the dynamics and the policy. We constrain the dynamics in this distribution to be equal to the true dynamics, which we do not need to actually know in practice but can simply sample in model-free fashion, and constrain the policy to some parameterized distribution. This defines the variational distribution $q(\tau)$ as

$$q(\tau) = p(\mathbf{s}_0) \prod_{t=0}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \qquad (3)$$

where $p(\mathbf{s}_0)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ are the true initial state distribution and dynamics, and $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is the parameterized policy that we wish to learn. We can fit this distribution by maximizing the evidence lower bound (ELBO):

$$\log p(\mathcal{O}_{0:T}) \geq -\mathrm{D}_{\mathrm{KL}}\left(q(\tau) \parallel p(\mathcal{O}_{0:T}, \tau)\right). \qquad (4)$$

Since the dynamics and initial state distributions in q and p match, it's straightforward to check that the divergence term simplifies to

$$J(\pi) = \mathbb{E}_{\tau \sim \rho_\pi(\tau)}\left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t) - \mathrm{D}_{\mathrm{KL}}\left(\pi(\,\cdot\,|\mathbf{s}_t) \parallel p(\,\cdot\,)\right)\right], \qquad (5)$$

which, if we choose a uniform action prior, is exactly the maximum entropy objective in Equation 1 up to a constant, and it can be optimized with any off-the-shelf entropy maximizing reinforcement learning algorithms (e.g. , (Nachum et al., 2017; Schulman et al., 2017a; Haarnoja et al., 2017; 2018)). Although the variational inference framework is not the only way to motivate the maximum entropy RL objective, it provides a convenient starting point for our method, which will augment the graphical model in Figure 1(a) with latent variables that can then be used to produce a hierarchy of policies, as discussed in the following section.

5. Learning Latent Space Policies

In this section, we discuss how the probabilistic view of RL can be employed in the construction of hierarchical policies, by augmenting the graphical model in Figure 1(a) with latent variables. We will also propose a particular way to parameterize the distribution over actions conditioned on these latent variables that is based on bijective transformations, which will provide us with a model amenable to stable and tractable training and the ability for higher levels in the hierarchy to fully invert the behavior of the lower layers, as
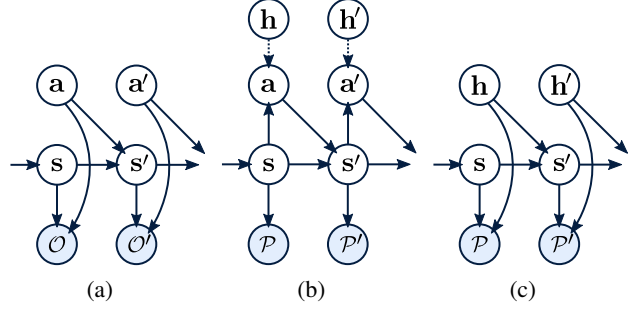

Figure 1. (a) The optimal control problem can be cast as an inference problem by considering a graphical model that consist of transition probabilities, action priors, and optimality variables. We can infer the optimal actions by conditioning on the optimality variables. (b) We can then train a latent variable policy to approximate the optimal actions, augment the graphical model with the policy's action distribution, and condition on a new set of optimality variables $\mathcal{P}_{0:T}$. Dashed line denotes a deterministic (invertible) edge. (c) By marginalizing out the actions $\mathbf{a}_t$, we are left with a new model that is structurally identical to the one in (a), where $\mathbf{h}_t$ has taken the role of the original actions.

we will discuss in Section 5.2. We will derive the method for two-layer hierarchies to simplify notation, but it can be easily generalized to arbitrarily deep hierarchies. In this work, we consider the bottom-up approach, where we first train a low-level policy, and then use it to provide a higher-level action space for a higher level policy that ideally can now solve an easier problem.

5.1. Latent Variable Policies for Hierarchical RL

We start constructing a hierarchy by defining a stochastic base policy as a latent variable model. In other words, we require the base policy to consist of two factors: a conditional action distribution $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_t)$, where $\mathbf{h}_t$ is a latent random variable, and a prior $p(\mathbf{h}_t)$. Actions can be sampled from this policy by first sampling $\mathbf{h}_t$ from the prior and then sampling an action conditioned on $\mathbf{h}_t$. Adding the latent variables $\mathbf{h}_t$ results in a new graphical model, which can now be conditioned on some new optimality variables $\mathcal{P}_t$ that can represent either the same task, or a different higher-level task, as shown in Figure 1(b). In this new graphical model, the base policy is integrated into the transition structure of the MDP, which now exposes a new, higher-level set of actions $\mathbf{h}_t$. Insofar as the base policy succeeds in solving the task, partially or completely, learning a related task with $\mathbf{h}_t$ as the action should be substantially easier.

This "policy-augmented" graphical model has a semantically identical interpretation as the original graphical model in Figure 1(a), where the combination of the transition model and the action conditional serves as a new (and likely easier) dynamical system. We can derive the dynamics

model for the combined system by marginalizing out the actions: $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{h}_t) = \int_{\mathcal{A}} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_t)d\mathbf{a}_t$, where $\mathbf{h}_t$ is a new, higher-level action and $p(\mathbf{h}_t)$ is its prior, as illustrated in Figure 1(c). In other words, the base policy shapes the underlying dynamics of the system, ideally making it more easily controllable by a higher level policy. We can now learn a higher level policy for the latents by conditioning on new optimality variables. We can repeat this process multiple times by integrating each new policy level into the dynamics and learning a new higher-level policy on top of the previous policy's latent space, thus constructing an arbitrarily deep hierarchical policy representation. We will refer to these policy layers as sub-policies in the following sections. Similar layerwise training has been studied in the context of generative modeling with deep belief networks and shown to improve optimization of deep architectures (Hinton & Salakhutdinov, 2006).

5.2. Practical Training of Latent Variable Policies

An essential choice in our method is the representation of sub-policies which, ideally, should be characterized by three properties. First, each sub-policy should be tractable. This is required, since we need to maximize the log-likelihood of good actions, which requires marginalizing out all latent variables. Second, the sub-policies should be expressive, so that they don't suppress the information flow from the higher levels to the lower levels. For example, a mixture of Gaussians as a sub-policy can only provide a limited number of behaviors for the higher levels, corresponding to the mixture elements, potentially crippling the ability of the higher layers to solve the task. Third, the conditional factor of each sub-policy should be deterministic, since the higher levels view it as a part of the environment, and additional implicit noise in the system can degrade their performance. Our approach to model the conditionals is based on bijective transformations of the latent variables into actions, which provides all of the aforementioned properties: the sub-policies are tractable, since marginalization reduces to single-point evaluation; they are expressive if represented via neural networks; and they are deterministic due to the bijective transformation. Specifically, we borrow from the recent advances in unsupervised learning based on real-valued non-volume preserving (real NVP) neural network transformations (Dinh et al., 2016). Our network differs from the original real NVP architecture in that we also condition the transformations on the current state or observation. Note that, even though the transformation from the latent to the action is bijective, it can depend on the observation in arbitrarily complex non-invertible ways, as we discuss in Section 6.1, providing our sub-policies with the requisite expressive power.

We can learn the parameters of these networks by utilizing the change of variables formula as discussed by Dinh

et al. (2016), and summarized here for completeness. Let $\mathbf{a}_t = f(\mathbf{h}_t; \mathbf{s}_t)$ be a bijective transformation, and let $\mathbf{h}_t \in \mathbb{R}^{|\mathcal{A}|}$ be a random variable and $p(\mathbf{h}_t)$ its prior density. It is possible to express the density of $\mathbf{a}_t$ in terms of the prior and the Jacobian of the transformation by employing the change of variable formula as

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = p(\mathbf{h}_t) \left| \det\left(\frac{df(\mathbf{h}_t; \mathbf{s}_t)}{d\mathbf{h}_t} \right) \right|^{-1}. \quad (6)$$

Dinh et al. (2016) propose a particular kind of bijective transformation, which has a triangular Jacobian, simplifying the computation of the determinant to a product of its diagonal elements. The exact structure of these transformations is outside of the scope of this work, and we refer the reader to (Dinh et al., 2016) for a more detailed description. We can easily chain these transformations to form multi-level policies, and we can train them end-to-end as a single policy or layerwise as a hierarchical policy. As a consequence, the policy representation is agnostic to whether or not it was trained as a single expressive latent-variable policy or as a hierarchical policy consisting of several sub-policies, allowing us to choose the training method that best suits the problem at hand without the need to redesign the policy topology each time from scratch. Next, we will discuss the different hierarchical training strategies in more detail.

5.3. Reward Functions for Policy Hierarchies

The simplest way to construct a hierarchy out of latent variable policies is to follow the procedure described in Section 5.1, where we train each layer in turn, then freeze its weights, and train a new layer that uses the lower layer's latent variables as an action space. In this procedure, each layer is trained on the same maximum entropy objective, and each layer simplifies the task for the layer above it. As we will show in Section 6, this procedure can provide substantial benefit on challenging and high-dimensional benchmark tasks.

However, for tasks that are more challenging, we can also naturally incorporate weak prior information into the training process by using underdefined heuristic reward functions for training the lower layers. In this approach, lower layers of the hierarchy are trained on reward functions that include shaping terms that elicit more desirable behaviors. For example, if we wish to learn a complex navigation task for a walking robot, the lower-layer objective might provide a reward for moving in any direction, while the higher layer is trained only on the primary objective of the task. Because our method uses bijective transformations, the higher layer can always undo any behavior in the lower layer if it is detrimental to task success, while lower layer objectives that correlate with task success can greatly simplify learning. Furthermore, since each layer still aims to maximize entropy, even a weak objective, such as a reward for motion

in any direction (e.g., the norm of the velocity), will produce motion in many different directions that is controllable by the lower layer's latent variables. In our experiments, we will demonstrate how this approach can be used to solve a goal navigation task for a simulated, quadrupedal robot.

5.4. Algorithm Summary

We summarize the proposed algorithm in Algorithm 1. The algorithm begins by operating on the low-level actions in the environment according to the unknown system dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, where we use $\mathbf{h}_t^{(0)} = \mathbf{a}_t$ to denote the lowest-layer actions. The algorithm is also provided with an ordered set of K reward functions $\mathcal{R}_i$, which can all represent the same task or different tasks, depending on the skill we want to learn: skills that naturally divide into primitive skills can benefit from specifying a different low-level objective, but for simpler tasks, such as locomotion, which do not naturally divide into primitives, we can train each sub-policy to optimize the same objective. In both cases, the last reward function $\mathcal{R}_{K-1}$ should correspond to the actual task we want to solve. The algorithm then chooses each $\mathcal{R}_i$ sequentially and learns a maximum entropy policy π_i, represented by an invertible transformation $f_i : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$ and a prior $p(\mathbf{h}_t^{(i)})$, to optimize the corresponding variational inference objective in Equation 5. Our proposed implementation uses soft actor-critic (Haarnoja et al., 2018) to optimize the policy due to its robustness and good sample-efficiency, although other entropy maximizing RL algorithms can also be used. After each iteration, we embed the newly learned transformation f_i into the environment, which produces a new system dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{h}_t^{(i+1)})$ that can be used by the next layer. As before, we do not need an analytic form of these dynamics, only the ability to sample from them, which allows our algorithm to operate in the fully model-free setting.

Algorithm 1 Latent Space Policy Learning

Input: True environment $p(\mathbf{s}_{t+1}|, \mathbf{s}_t, \mathbf{h}_t^{(0)})$, where $\mathbf{h}_t^{(0)}$
   corresponds to the physical actions $\mathbf{a}_t$.
Input: Reward specifications $\{\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_{K-1}\}$.
   for $i = 0$ to $K - 1$ **do**
      Initialize the weights of layer f_i.
      Learn the weights of f_i so that $\mathbf{h}_t^{(i)} = f_i(\mathbf{h}_t^{(i+1)}; \mathbf{s}_t)$,
         where $\mathbf{h}_t^{(i+1)} \sim p(\mathbf{h}_t^{(i+1)})$, optimizes $\mathcal{R}_i$
         on $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{h}_t^{(i)})$.
      Embed the new layer f_i into the environment:
         $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{h}_t^{(i+1)}) \leftarrow p(\mathbf{s}_{t+1}|\mathbf{s}_t, f_i(\mathbf{h}_t^{(i+1)}; \mathbf{s}_t))$.
   end for
Output: A hierarchical policy $f = f_0 \circ f_1 \circ ... \circ f_{K-1}$.
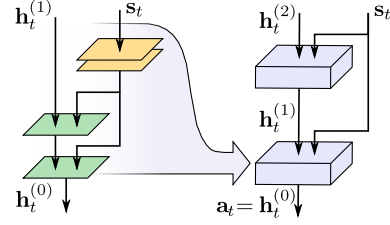

Figure 2. Our hierarchical policy consist of two levels (right diagram) that take in the observation and a latent vector from the previous level and outputs a latent vector to the next level. Diagram on the left shows the internal structure of each of the policy levels. The latent vector that is passed from the higher level is fed through two invertible coupling layers (green) (Dinh et al., 2016), which we condition on an observation embedding (yellow). Note that the path from the observation to the output does not need to be bijective, and therefore the observation embeddings can be represented with an arbitrary neural network, which in our case consists of two fully connected layers.

6. Experiments

Our experiments were conducted on several continuous control benchmark tasks from the OpenAI Gym benchmark suite (Brockman et al., 2016). The aim of our experiments was to answer the following questions: (1) How well does our latent space policy learning method compare to prior reinforcement learning algorithms? (2) Can we attain improved performance from adding additional latent variable policy layers to a hierarchy, especially on challenging, high-dimensional tasks? (3) Can we solve more complex tasks by providing simple heuristic shaping to lower layers of the hierarchy, while the higher layers optimize the original task reward? Videos of our experiments are available online[1].

6.1. Policy Architecture

In our experiments, we used both single-level policies and hierarchical policies composed of two sub-policies as shown on the right in Figure 2. Each sub-policy has an identical structure, as depicted on the left in Figure 2. A sub-policy is constructed from two *coupling layers* that are connected using the alternating pattern described in (Dinh et al., 2016). Our implementation differs from Dinh et al. (2016) only in that we condition the coupling layers on the observations, via an embedding performed by a two-layer fully-connected network. In practice, we concatenate the embedding vector with the latent input to each coupling layer. Note that our method requires only the path from the input latent to the output to be invertible, and the output can depend on the observation in arbitrarily complex ways. We have released our code for reproducibility.[2]

---

[1]https://sites.google.com/view/latent-space-deep-rl
[2]https://github.com/haarnoja/sac

(a) Swimmer (rllab)   (b) Hopper-v1

(c) Walker2d-v1   (d) HalfCheetah-v1
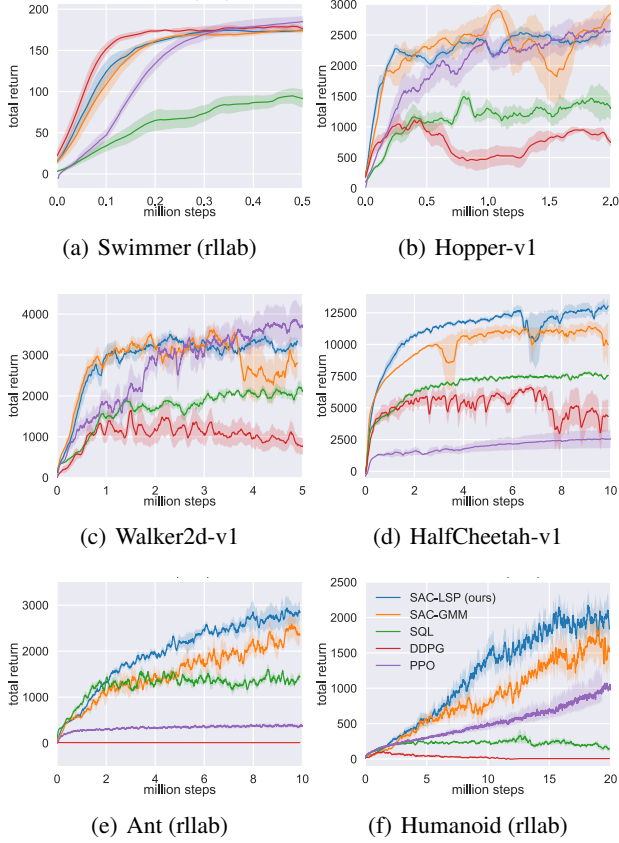

(e) Ant (rllab)   (f) Humanoid (rllab)

Figure 3. Training curves for continuous control benchmarks. Thick lines correspond to mean performance, and shaded regions show standard deviations of five random seeds. Our method (SAC-LSP) attains state-of-the-art performance across all tasks.

6.2. Benchmark Tasks with Single-Level Policies

We compare our method (SAC-LSP) to two commonly used RL algorithms: proximal policy optimization (PPO) (Schulman et al., 2017b), a commonly used policy gradient algorithm, and deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), which is a sample efficient off-policy method. We also include two recent algorithm that learn maximum entropy policies: soft Q-learning (SQL) (Haarnoja et al., 2017), which also learns a sampling network as part of the model, represented by an implicit density model, and soft actor-critic (Haarnoja et al., 2018), which uses a Gaussian mixture model policy (SAC-GMM). Note that the benchmark tasks compare the total expected return, but the entropy maximizing algorithms optimize a slightly different objective, so this comparison slightly favors PPO and DDPG, which optimize the benchmark objective directly. Another difference between the two classes of algorithms is that the maximum entropy policies are stochastic at test time, while DDPG is deterministic and PPO typically converges to nearly deterministic policies. For SAC-GMM, we execute an approximate maximum a posteriori action by choosing the mean of the mixture com-
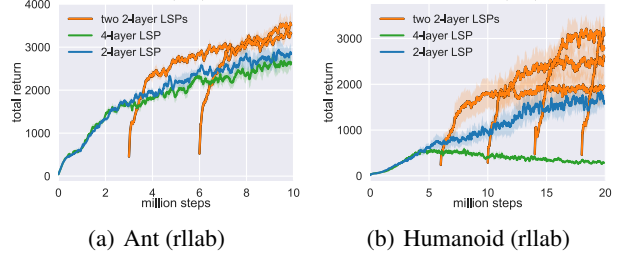

(a) Ant (rllab)   (b) Humanoid (rllab)

Figure 4. (a, b) We trained two-level policies for the most challenging benchmark tasks, Ant and Humanoid, by first training a single level policy (blue) and then freezing it and adding a second policy level. We repeated this procedure by starting training of the top-level policy at multiple points in time (orange). We also trained a single policy with four invertible layers end-to-end (green) for comparison. In each case, stagewise training of two levels yields the best performance.

ponent that has the highest Q-value at test time, but for SQL and our method, which both can represent an arbitrarily complex posterior distribution, we simply sample from the stochastic policy.

Our results on the benchmark tasks show that our policy representation generally performs on par or better than all of the tested prior methods, both in terms of efficiency and final return (Figure 3), especially on the more challenging and high-dimensional tasks, such as Humanoid. These results indicate that our policy representation can accelerate learning, particularly in challenging environments with high-dimensional actions and observations.

6.3. Multi-Level Policies

In this section, we evaluate the performance of our approach when we compose multiple latent variable policies into a hierarchy. In the first experiment, we train a single-level base policy on the most challenging standard benchmarks, Ant and Humanoid. We then freeze the weights of the base policy, and learn another policy level that uses the latent variables of the first policy as its action space. Intuitively, each layer in such a hierarchy attempts to solve the task to the best of its ability, providing an easier problem for the layer above. In Figure 4(a) and Figure 4(b), we show the training curves for Ant and Humanoid, where the blue curve corresponds to the base policy and orange curves show the performance after we freeze the base policy weights and optimize a second-level policy. The different orange curves correspond to the addition of the second layer after a different numbers of training steps. In each case, the two-level policy can outperform a single level policy by a large margin. The performance boost is more prominent if we train the base policy longer. Note that the base policy corresponds to a single-level policy in Figure 3, and already learns more efficiently than prior methods. We also compare to a single, more expressive policy (green) that consists of four invertible layers, which has a similar number of parameters to a

stacked two-level policy, but trained end-to-end as oppose to stagewise. This single four-layer policy performs comparably (Ant) or worse (Humanoid) than a single, two-layer policy (blue), indicating that the benefit of the two-level hierarchy is not just in the increased expressivity of the policy, but that the stagewise training procedure plays an important role in improving performance.

In our second experiment, we study how our method can be used to build hierarchical policies for complex tasks that require compound skills. The particular task we consider requires Ant to navigate through a simple maze (Figure 5(a)) with a sparse, binary reward for reaching the goals (shown in green). To solve this task, we can construct a hierarchy, where the lower layer is trained to acquire a general locomotion skill simply by providing a reward for maximizing velocity, regardless of direction. This pretraining phase can be conducted in a simpler, open environment where the agent can move freely. This provides a small amount of domain knowledge, but this type of domain knowledge is much easier to specify than true intermediate goals or modulation (Heess et al., 2016; Kulkarni et al., 2016), and the entropy maximization term in the objective automatically causes the low-level policy to learn a range of locomotion skills for various directions. The higher-level policy is then provided the standard task reward. Because the lower-level policy is invertible, the higher-level policy can still solve the task however it needs to, potentially even by fully undoing the behavior of the lower-level policy. However, the results suggest that the lower-level policy does indeed substantially simplify the problem, resulting in both rapid learning and good final performance.

In Figure 5(b), we compare our approach (blue) to four single-level baselines that either learn a policy from scratch or fine-tune a pretrained policy. The pretraining phase (4 million steps) is not included in the learning curves, since the same base policies were reused across multiple tasks, corresponding to the three difference goal locations shown in Figure 5(a). With task reward only, training a policy from scratch (red) failed to solve the task due to lack of structured exploration, whereas fine-tuning a pretrained policy (brown) that already knows how to move around and could occasionally find the way to the goal was able to slowly learn this task. We also tried improving exploration by augmenting the objective with a motion reward, provided as a shaping term for the entire policy. In this case, a policy trained from scratch (purple) learned slowly, as it first needed to acquire a locomotion skill, but was able to eventually solve the task, while fine-tuning a pretrained policy resulted in much faster learning (pink). However, in both cases, adding motion as a shaping term prevents the policy from converging on an optimal solution, since the shaping alters the task. This manifests as residual error at convergence. On the other hand, our method (blue) can make use
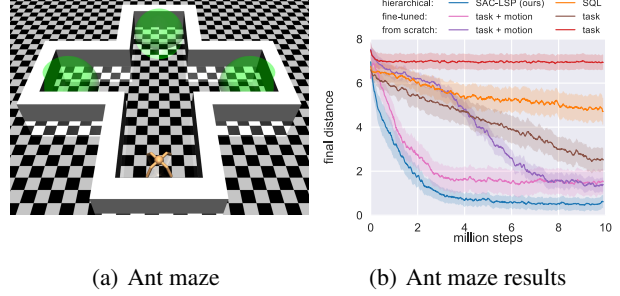

(a) Ant maze   (b) Ant maze results

Figure 5. (a) We trained Ant to navigate through a simple maze to three different goal location shown in green. (b) We first trained a low-level policy with a motion reward in a pretraining environment not including the walls, then fixed the policy and trained another policy level with a target reward (blue). We compare our method to learning a single policy from scratch or fine-tuning the pretrained policy using only the task reward or a combination of task and motion rewards. We also applied our method to soft Q-learning.

of the pretrained locomotion skills while optimizing for the task reward directly, resulting in faster learning and better final performance. Note that our method converges to a solution where the final distance to the goal is more than four times smaller than for the next best method, which finetunes with a shaped reward. We also applied our method to soft Q-learning (yellow), which also trains latent space policies, but we found it to learn substantially slower than SAC-LSP.

7. Discussion and Future Work

We presented a method for training policies with latent variables. Our reinforcement learning algorithm not only compares favorably to state-of-the-art algorithms on standard benchmark tasks, but also provides an appealing avenue for constructing hierarchical policies: higher levels in the hierarchy can directly make use of the latent space of the lower levels as their action space, which allows us to train the entire hierarchy in a layerwise fashion. This approach to hierarchical reinforcement learning has a number of conceptual and practical benefits. First, each layer in the hierarchy can be trained with exactly the same algorithm. Second, by using an invertible mapping from latent variables to actions, each layer becomes invertible, which means that the higher layer can always perfectly invert any behavior of the lower layer. This makes it possible to train lower layers on heuristic shaping rewards, while higher layers can still optimize task-specific rewards with good asymptotic performance. Our method has a natural interpretation as an iterative procedure for constructing graphical models that gradually simplify the task dynamics. Our approach can be extended to enable different layers to operate at different temporal scales, which would provide for extensions into domains with temporally delayed rewards and multi-stage tasks.

Acknowledgments

References

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1726–1734, 2017.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Daniel, C., Neumann, G., and Peters, J. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pp. 273–281, 2012.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

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

Florensa, C., Duan, Y., and P., A. Stochastic neural networks for hierarchical reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Fox, R., Pakman, A., and Tishby, N. Taming the noise in reinforcement learning via soft updates. In *Conference on Uncertainty in Artificial Intelligence*, 2016.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, pp. 1352–1361, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *Conference on Learning Representations (ICLR)*, 2018.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science*, 313 (5786):504–507, 2006.

Kappen, H. J. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory And Experiment*, 2005(11):P11011, 2005.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3675–3683, 2016.

Kupcsik, A. G., Deisenroth, M. P., Peters, J., and Neumann, G. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1401–1407, 2013.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521:436–444, May 2015.

Levine, S. *Motor skill learning with local trajectory methods*. PhD thesis, Stanford University, 2014.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

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

MacAlpine, P. and Stone, P. Overlapping layered learning. *Artificial Intelligence*, 254:21–43, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2772–2782, 2017.

Neumann, G. Variational inference for policy search in changing situations. In *International Conference on Machine Learning (ICML)*, pp. 817–824, 2011.

Peters, J., Mülling, K., and Altun, Y. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1607–1612, 2010.

Rawlik, K., Toussaint, M., and Vijayakumar, S. On stochastic optimal control and reinforcement learning by approximate inference. *Robotics: Science and Systems (RSS)*, 2012.

Saxe, A. M., Earle, A. C., and Rosman, B. S. Hierarchy through composition with multitask LMDPs. *Proceedings of Machine Learning Research*, 2017.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning (ICML)*, pp. 1312–1320, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897, 2015.

Schulman, J., Abbeel, P., and Chen, X. Equivalence between policy gradients and soft Q-learning. *arXiv preprint arXiv:1704.06440*, 2017a.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 0028-0836. Article.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 3, pp. 6, 2017.

Todorov, E. Linearly-solvable Markov decision problems. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1369–1376. MIT Press, 2007.

Toussaint, M. Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)*, pp. 1049–1056. ACM, 2009.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

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

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1433–1438, 2008.

A. Hyperparameters

A.1. Common Parameters

We use the following parameters for LSP policies throughout the experiments. The algorithm uses a replay pool of one million samples, and the training is delayed until at least 1000 samples have been collected to the pool. Each training iteration consists of 1000 environments time steps, and all the networks (value functions, policy scale/translation, and observation embedding network) are trained at every time step. Every training batch has a size of 128. The value function networks and the embedding network are all neural networks comprised of two hidden layers, with 128 ReLU units at each hidden layer. The dimension of the observation embedding is equal to two times the number of action dimensions. The scale and translation neural networks used in the real NVP bijector both have one hidden layer consisting of number of ReLU units equal to the number of action dimensions. All the network parameters are updated using Adam optimizer with learning rate $3 \cdot 10^{-4}$.

Table 1 lists the common parameters used for the LSP-policy, and Table 2 lists the parameters that varied across the environments.

Table 1. Shared parameters for benchmark tasks

Parameter	Value
learning rate	$3 \cdot 10^{-4}$
batch size	128
discount	0.99
target smoothing coefficient	10^{-2}
maximum path length	10^3
replay pool size	10^6
hidden layers (Q, V, embedding)	2
hidden units per layer (Q, V, embedding)	128
policy coupling layers	2

A.2. High-Level Policies

All the low-level policies in hierarchical cases (Figures 5(b), 4(a), 4(b)) are trained using the same parameters used for the corresponding benchmark environment. All the high-level policies use Gaussian action prior. For the Ant maze task, the latent sample of the high-level policy is sampled once in the beginning of the rollout and kept fixed until the next one. The same high-level action is kept fixed over three environment steps. Otherwise, all the policy parameters for the high-level policies are equal to the benchmark parameters.

The environments used for training the low-level policies are otherwise equal to the benchmark environments, except for their reward function, which is modified to yield velocity based reward in any direction on the xy-plane, in contrast to just positive x-direction in the benchmark tasks. In the Ant maze environment, the agent receives a reward of 1000 upon reaching the goal and 0 otherwise. In particular, no velocity reward nor any control costs are awarded to the agent. The environment terminates after the agent reaches the goal.

Table 2. Environment specific parameters for benchmark tasks

Parameter	Swimmer (rllab)	Hopper-v1	Walker2d-v1	HalfCheetah-v1	Ant (rllab)	Humanoid (rllab)
action dimensions	2	3	6	6	8	21
reward scale	100	1	3	1	3	3
observation embedding dimension	4	6	12	12	16	42
scale/translation hidden units	2	3	6	6	8	21