Learning Robot Skill Embeddings

Karol Hausman*
Department of Computer Science, University of Southern California
hausman@usc.edu

Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, Martin Riedmiller
DeepMind
{springenberg,ziyu,heess,riedmiller}@google.com

Abstract

We present a method for reinforcement learning of closely related skills that are parameterized via a skill embedding space. We learn such skills by taking advantage of latent variables and exploiting a connection between reinforcement learning and variational inference. The main contribution of our work is an entropy-regularized policy gradient formulation for hierarchical policies, and an associated, data-efficient and robust off-policy gradient algorithm based on stochastic value gradients. We demonstrate the effectiveness of our method on several simulated robotic manipulation tasks. We find that our method allows for discovery of multiple solutions and is capable of learning the minimum number of distinct skills that are necessary to solve a given set of tasks. In addition, our results indicate that the hereby proposed technique can interpolate and/or sequence previously learned skills in order to accomplish more complex tasks, even in the presence of sparse rewards.

1 Introduction

Recent years have seen great progress in methods for reinforcement learning with rich function approximators, aka "deep reinforcement learning" (DRL) [25, 35, 20]. In the field of robotics, DRL holds the promise of automatically learning flexible behaviors end-to-end while dealing with high-dimensional, multi-modal sensor streams [1].

Despite this recent progress, the predominant paradigm remains, however, to learn solutions from scratch for every task presented to the RL algorithm. Not only is this data-inefficient and constrains the difficulty of the tasks that can be solved, but it also limits the versatility and adaptivity of the systems that can be built. This is by no means a novel insight and there have been many attempts to address this issue (e.g. [7, 33, 8, 37]). Nevertheless, the effective discovery, representation, and reuse of skills remains an open research question.

We aim to take a step towards this goal. Our method learns manipulation skills that are continuously parameterized in an embedding space. We show how we can take advantage of these skills for rapidly solving new tasks, effectively by solving the control problem in the embedding space rather than the raw action space.

Our formulation draws on a connection between entropy-regularized reinforcement learning and variational inference (VI) (that is well established in the literature [38, 39, 42, 31, 29, 19, 10]) and is a principled and general scheme for learning hierarchical stochastic policies. We show how stochastic latent variables can be meaningfully incorporated into policies by treating them in the same way as auxiliary variables in parametric variational approximations in inference ([34, 22, 30]).

*This work was carried out during an internship at DeepMind.

The resulting policies can model complex correlation structure and multi-modality in action space. We represent the skill embedding via such latent variables and find that this view naturally leads to an information-theoretic regularization which ensures that the learned skills are versatile and the embedding space is well formed.

We demonstrate the effectiveness of our method on several simulated robotic manipulation tasks. We find that our method allows for the discovery of multiple solutions and is capable of learning the minimum number of distinct skills that are necessary to solve a given set of tasks. Our results indicate that the hereby proposed technique can interpolate and/or sequence previously learned skills in order to accomplish more complex tasks, even in the presence of sparse rewards. The video of our experiments is available at: `https://goo.gl/FbvPGB`.

2 Related Work

In the space of multi-task reinforcement learning with neural networks, Teh et al. [37] propose a framework that allows sharing of knowledge across tasks via a task agnostic prior. Similarly, Cabi et al. [4] make use of off-policy learning to learn about a large number of different tasks while following a main task. Denil et al. [6] and Devin et al. [7] propose architectures that can be reconfigured to solve a variety of tasks, and Finn et al. [8] use meta-learning to acquire skills that can be fine-tuned effectively. Sequential learning and the need to retain previously learned skills has also been the focus of a number of researchers (e.g. [18] and [33]). In this work, we present a method that learns an explicit skill embedding space in a multi-task setting and is complementary to these works.

Our formulation draws on a connection between entropy-regularized reinforcement learning and variational inference (VI) (e.g. [38, 39, 42, 31, 29, 19, 10]). In particular, it considers formulations with auxiliary latent variables, a topic studied in the VI literature (e.g. [3, 34, 30, 22]) but not fully explored in the context of RL. The notion of latent variables in policies has been explored e.g. by controllers [15] or options [2]. Their main limitation is the lack of a principled approach to avoid a collapse of the latent distribution to a single mode. The auxiliary variable perspective introduces an information-theoretic regularizer that helps the inference model by producing more versatile behaviors. Learning versatile skills has been explored by Haarnoja et al. [12] and Schulman et al. [36]. In particular, Haarnoja et al. [12] learn energy-based, maximum entropy policies via soft Q-learning. Our approach similarly uses entropy-regularized reinforcement learning and latent variables but differs in the algorithmic framework. Similar hierarchical approaches have also been studied in the work combining RL with imitation learning [40, 24].

The works that are most closely related to this paper are [9, 27, 11] and [13, 21]. They use the same bound that arises in our treatment of the latent variables. Hausman et al. [13] uses it to learn structure from demonstrations, while Mohamed & Rezende [27], Gregor et al. [11] use mutual information as an intrinsic reward for option discovery. Florensa et al. [9] follows a similar paradigm of pre-training stochastic neural network policies, which are then used to learn a new task in an on-policy setup. This approach can be viewed as a special case of the method introduced in this paper, where the skill embedding distribution is a fixed uniform distribution and an on-policy method is used to optimize the regularized objective. In contrast, our method is able to learn the skill embedding distributions, which enables interpolation between different skills as well as discovering the number of distinct skills necessary to accomplish a set of tasks. In addition, we extend our method to a more sample-efficient off-policy setup, which is important for potential applications of this method to real world environments.

3 Learning Versatile Skills

Before we introduce our method for learning a latent skill embedding space, it is instructive to identify the exact desiderata that we impose on the acquired skills (and thus the embedding space parameterizing them). A detailed explanation of how these goals align with recent trends in the literature is given in Section 2.

As stated in the introduction, the general goal of our method is to re-use skills learned for an initial set of tasks to speed up – or in some cases even enable – learning difficult target tasks in a transfer learning setting. We are thus interested in the following properties for the initially learned skills:

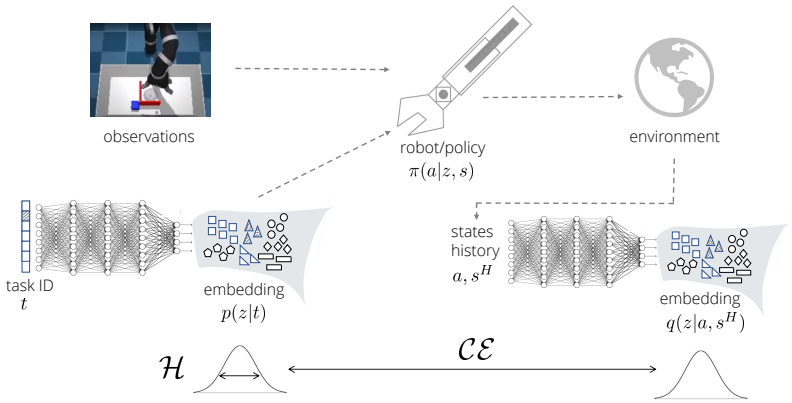

Figure 1: Schematics of our approach. We train the agent in a multi-task setup, where the task id is given as a one-hot input to the embedding network (bottom-left). The embedding network generates an embedding distribution that is sampled and concatenated with the current observation to serve as an input to the policy. After interaction with the environment, a segment of states is collected and fed into the inference network (bottom-right). The inference network is trained to classify what embedding vector the segment of states was generated from.

i) generality: We desire an embedding space, in which solutions to different, potentially orthogonal, tasks can be represented; i.e. tasks such as lifting a block or pushing it through an obstacle course should both be jointly learnable by our approach.

ii) versatility: We aim to learn a skill embedding space, in which different embedding vectors that are "close" to each other in the embedding space correspond to distinct solutions to the same task.

iii) identifiability: Given the state and action trace of an executed skill, it should be possible to identify the embedding vector that gave rise to the solution. This property would allow us to re-purpose the embedding space for solving new tasks by picking a sequence of embedding vectors.

Intuitively, the properties i)-ii) of generality and versatility can be understood as: "we hope to cover as much of the skill embedding space as possible with different clusters of task solutions, within each of which multiple solutions to the same task are represented". Property iii) intuitively helps us to: "derive a new skill by re-combining a diversified library of existing skills".

3.1 Policy Learning via a Variational Bound on Entropy Regularized RL

To learn the skill-embedding we assume to have access to a set of initial tasks $\mathcal{T} = [1, \ldots, T]$ with accompanying, per-task, reward functions $r_t(s, a)$, which could be comprised of different environments, variable robot dynamics, reward functions, etc.. At training time, we provide access to the task id $t \in \mathcal{T}$ (indicating which task the agent is operating in) to our RL agent. In practice – to obtain data from all training tasks for learning – we draw a task and its id randomly from the set of tasks $\mathcal{T}$ at the beginning of each episode and execute the agents current policy $\pi(a|s, t)$ in it. A conceptual diagram presenting our approach is depicted in Appendix in Fig. 1.

For our policy to learn a diverse set of skills instead of just T separate solutions (one per task), we endow it with a task-conditional latent variable z. With this latent variable, which we also refer to as "skill embedding", the policy is able to represent a distribution over skills for each task and to share these across tasks. In the simplest case, this latent variable could be resampled at every timestep and the state-task conditional policy would be defined as $\pi(a|s, t) = \int \pi(a|z, s, t)p(z|t)dz$. One simple choice would be to let $z \in 1, \ldots K$, in which case the policy would correspond to a mixture of K subpolicies.

Introducing a latent variable facilitates the representation of several alternative solutions but it does not mean that several alternative solutions will be learned. It is easy to see that the expected reward objective does not directly encourage such behavior. To achieve this, we formulate our objective as

an entropy regularized RL problem, i.e. we maximize:

$$\max_{\pi} \mathbb{E}_{\pi, p_0, t\in\mathcal{T}}\Big[\sum_{i=0}^{\infty}\gamma^i\big(r_t(s_i,a_i)+\alpha\mathcal{H}[\pi(a_i|s_i,t)]\big)\Big|a_i\sim\pi(\cdot|s,t), s_{i+1}\sim p(s_{i+1}|a_i,s_i)\Big], \quad (1)$$

where $p_0(s_0)$ is the initial state distribution, α is a weighting term – trading the arbitrarily scaled reward against the entropy – and we can define $R(a,s,t) = \mathbb{E}_{\pi}[\sum_{i=0}^{\infty}\gamma^i r_t(s_i,a_i)|s_0 = s, a_i \sim \pi(\cdot|s,t)]$ to denote the expected return for task t (under policy π) when starting from state s and taking action a. The entropy regularization term is defined as: $\mathcal{H}[\pi(a|s,t)] = \mathbb{E}_{\pi}[-\log\pi(a|s,t)]$. It is worth noting that this is very similar to the "entropy regularization" conventionally applied in many policy gradient schemes (e.g. [41, 26]) but with the critical difference that it takes into account not just the entropy of the current but also of future actions.

To apply this entropy regularization to our setting, i.e. in the presence of latent variables, extra machinery is necessary since the entropy term becomes intractable for most distributions of interest. Borrowing from the toolkit of variational inference and applying the bound from [3], we can construct a lower bound on the entropy term from Equation (1) as (see Appendix A.3 for details):

$$\begin{aligned}
&\mathbb{E}_{\pi}[-\log\pi(a|s,t)] \\
&\geq \mathbb{E}_{\pi(a,z|s,t)}\Big[\log\Big(\frac{q(z|a,s,t)}{\pi(a,z|s,t)}\Big)\Big] \\
&= -\mathbb{E}_{\pi_\theta(a|s,t)}\Big[\mathcal{CE}\big[p(z|a,s,t)\|q_\psi(z|a,s)\big]\Big] + \mathcal{H}[p_\phi(z|t)] + \mathbb{E}_{p_\phi(z|t)}\Big[\mathcal{H}[\pi_\theta(a|s,z)]\Big],
\end{aligned} \quad (2)$$

where $q(z|a,s,t)$ is a variational inference distribution that we are free to choose, and $\mathcal{CE}$ denotes the cross entropy (CE). Note that although $p(z|a,s,t)$ is intractable, a sample based evaluation of the CE term is possible:

$$\mathbb{E}_{\pi_\theta(a|s,t)}\Big[\mathcal{CE}\big[p(z|a,s,t)\|q_\psi(z|a,s)\big]\Big] = \mathbb{E}_{\pi_\theta(a,z|s,t)}\Big[-\log q_\psi(z|a,s)\Big].$$

This bound holds for any q. We choose q such that it complies with our desired property of **identifiability** (cf. Section 3): we avoid conditioning q on the task id t to ensure that a given trajectory alone will allow us to identify its embedding. The above variational bound is not only valid on a single state, but can also be easily extended to a short trajectory segment of states $s_i^H = [s_{i-H},\dots,s_i]$, where H is the segment length. We thus use the variational distribution $q_\psi(z|a,s_i^H)$ – parameterized via a neural network, which we refer to as the inference network, with parameters ψ. We also represent the policy $\pi_\theta(a|s,z)$ and the embedding distribution $p_\phi(z|t)$ using neural networks – with parameters θ and ϕ – and refer to them as policy and embedding networks respectively. The above formulation is for a single time-step; we describe a more general formulation in the Appendix (Section A.4).

The bound meets the desiderata from Section 3: it maximizes the entropy of the embedding given the task $\mathcal{H}(p(z|t))$ and the entropy of the policy conditioned on the embedding $\mathbb{E}_{p(z|t)}\mathcal{H}(\pi(a|s,z))$ (thus, aiming to cover the embedding space with different skill clusters). The negative CE encourages different embedding vectors z to have different effects in terms of executed actions and visited states: Intuitively, it will be high when we can predict z from the resulting a, s^H. The first two terms in our bound also arise from the bound on the mutual information presented in [9]. We refer to the related work section for an in-depth discussion. We highlight that the above derivation also holds for the case where the task id is constant (or simply omitted) resulting in a bound for learning a latent embedding space encouraging the development of diverse solutions to a single task.

Inserting Equation (2) into our objective from Equation (1) yields the variational bound

$$\begin{aligned}
&L(\theta,\phi,\psi) = \mathbb{E}_{\substack{\pi_\theta(a,z|s,t)\\ t\in\mathcal{T}}}\left[\sum_{i=0}^{\infty}\gamma^i\hat{r}(s_i,a_i,z,t)\Big|s_{i+1}\sim p(s_{i+1}|a_i,s_i))\right] + \alpha_1\mathbb{E}_{t\in\mathcal{T}}\Big[\mathcal{H}[p_\phi(z|t)]\Big], \\
&\text{where } \hat{r}(s_i,a_i,z,t) = \Big[r_t(s_i,a_i) + \alpha_2\log q_\psi(z|a_i,s_i^H) + \alpha_3\mathcal{H}[\pi_\theta(a|s_i,z)]\Big],
\end{aligned} \quad (3)$$

with split entropy weighting terms $\alpha = \alpha_1+\alpha_2+\alpha_3$. Note that $\mathbb{E}_{t\in\mathcal{T}}\Big[\mathcal{H}[p_\phi(z|t)]\Big]$ does not depend on the trajectory.

4 Learning an Embedding for Versatile Skills in an Off-Policy Setting

While the objective presented in Equation (3) could be optimized directly in an on-policy setting (similar to [9]), our focus in this paper is on obtaining a data-efficient, off-policy, algorithm. The bound presented in the previous section requires environment interaction to estimate the discounted sums presented in the first three terms of Equation (3). These terms can, however, also be estimated efficiently from previously gathered data by learning a Q-value function[2], yielding an off-policy algorithm.

We assume the availability of a replay buffer $\mathcal{B}$ (containing full trajectory execution traces icluding states, actions, task id and reward), that is incrementally filled during training (see the appendix for further details). In conjunction with the trajectory traces, we also store the probabilities of each selected action and denote them with the behavior policy probability $b(a|z,s,t)$ as well as the behaviour probabilities of the embedding $b(z|t)$.

Given this replay data, we formulate the off-policy perspective of our algorithm. We start with the notion of a *lower-bound Q-function* that depends on both state s and action a and is conditioned on both, the embedding z and the task id t. It encapsulates all time dependent terms from Equation (3) and can be recursively defined as:

$$Q^{\pi}(s_i, a_i; z, t) = \hat{r}(s_i, a_i, z, t) + \gamma \mathbb{E}_{p(s_{i+1}|a_i,s_i)}[Q^{\pi}(s_{i+1}, a_{i+1}; z, t)]. \tag{4}$$

To learn a parametric representation of $Q^{\pi}_{\varphi}(s, a, ; z, t)$, we turn to the standard tools for policy evaluation from the RL literature. Specifically, we make use of the recent Retrace algorithm from [28]. We refer to Section A.2 for a more detailed description of this step. Equipped with this Q-function, we can update the policy and embedding network parameters without requiring additional environment interaction (using only data from the replay buffer) by optimizing the following off-policy objective:

$$\hat{L}(\theta, \phi) = \mathbb{E}_{\substack{\pi_\theta(a|z,s) \\ p_\phi(z|t) \\ s,t \in \mathcal{B}}} \Bigg[Q^{\pi}_{\varphi}(s, a, z) \Bigg] + \mathbb{E}_{t \in \mathcal{T}} \Big[\mathcal{H}[p_\phi(z|t)] \Big], \tag{5}$$

which can be readily obtained by inserting Q^{π}_{φ} into Equation (3). To minimize this objective via gradient descent, we draw further inspiration from recent successes in variational inference and directly use the pathwise derivative of Q^{π}_{φ} w.r.t. the network parameters by using the reparametrization trick [17, 32]. This method has previously been adapted for off-policy RL in the framework of stochastic value gradient algorithms [14] and was found to yield low-variance estimates.

For the inference network $q_\psi(z|a, s^H)$, minimizing equation (3) amounts to supervised learning, maximizing:

$$\hat{L}(\psi) = \mathbb{E}_{\substack{\pi_\theta(a,z|s,t) \\ t \in \mathcal{T}}} \Bigg[\sum_{i=0}^{\infty} \gamma^i \log q_\psi(z|a, s^H) \Big| s_{i+1} \sim p_\pi(s_{i+1}|a_i, s_i) \Bigg], \tag{6}$$

which requires sampling new trajectories to acquire target embeddings consistent with the current policy and embedding network. We found that simply re-using sampled trajectory snippets from the replay buffer works well empirically; allowing us to update all network parameters at the same time. Together with our choice for learning a Q-function, this results in a sample efficient algorithm. We refer to Section A.5.1 for the derivation of the stochastic value gradient of Equation (5).

5 Learning to Control the Previously-Learned Embedding

Once the skill-embedding is learned using the described multi-task setup, we utilize it to learn a new skill. There are multiple possibilities to employ the skill-embedding in such a scenarion including fine-tuning the entire policy or learning only a new mapping to the embedding space (modulating the lower level policies). In this work, we decide to focus on the latter: To adapt to a new task we freeze the policy network and only learn a new state-embedding mapping $z = f_\vartheta(x)$ via a neural network f_ϑ (parameterized by parameters ϑ). In other words, we only allow the network to learn how to modulate and interpolate between the already-learned skills, but we do not allow to change the underlying policies.

[2]From the perspective of variational inference, from which we are drawing inspiration in this paper, such a Q function can be interpreted as an amortized inference network estimating a log-likelihood term.

6 Experimental Results

Our experiments aim to answer the following questions: (1) Can our method learn versatile skills? (2) Can it determine how many distinct skills are necessary to accomplish a set of tasks? (3) Can we use the learned skill embedding for control in an unseen scenario? (4) Is it important for the skills to be versatile to use their embedding for control? (5) Is it more efficient to use the learned embedding rather than to learn to solve a task from scratch? We evaluate our approach in two domains in simulation: a point mass task to easily visualize different properties of our method and a set of challenging robot manipulation tasks. Our implementation uses 16 asynchronous workers interacting with the environment, and synchronous updates utilizing the replay buffer data.

6.1 Didactic Example: Multi-Goal Point Mass Task with Sparse Rewards

The didactic example consists of a point mass that is rewarded for being in a goal region. In particular, we consider a case where there are four goals, that are located around the initial location (see Fig. 2 left and middle) and each of them is equally important (the agent obtains the same reward at each location) for a single task ($T = 1$). This leads to a situation where there exist multiple optimal policies for a single task. In addition, this task is challenging due to the sparsity of the rewards – as soon as one solution is discovered, it is difficult to keep exploring other goals. Due to these challenges, most existing DRL approaches would be content with finding a single solution. For this experiment, we consider both, a Gaussian embedding space as well as a multivariate Bernoulli distribution (which we expect to be more likely to capture the multi-modality of the solutions).

The left part of Fig. 2 presents the versatility of the solutions when using the multivariate Bernoulli (left) and Gaussian (middle) embedding. The multivariate Bernoulli distribution is able to discover all four solutions, whereas the Gaussian embedding focuses on discovering different trajectories that lead to only two of the goals.

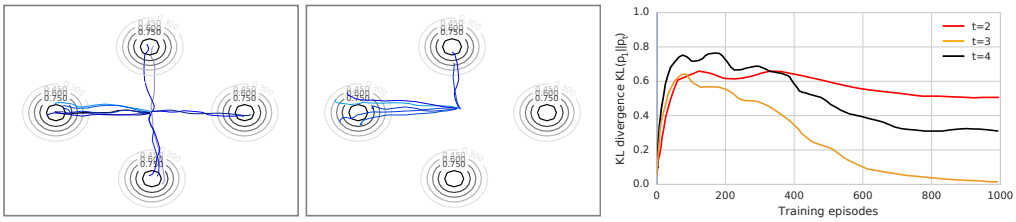

Figure 2: Left, middle: resulting trajectories that were generated by different distributions used for the skill-embedding space: multivariate Bernoulli (left), Gaussian (middle). The contours depict the reward gained by the agent. Note that there is no reward outside the goal region. Right: KL-divergence between the embedding distributions produced by task 1 and other three tasks. Task 1 and 3 have different task ids but are they are exactly the same tasks. Our method is able to discover that task 1 and 3 can be covered by the same embedding, which corresponds to the minimal KL-divergence between their embeddings.

In order to evaluate whether our method can determine the number of distinct skills that are necessary to accomplish a set of tasks, we conduct the following experiment. We set the number of task to four ($T = 4$) but we set two of the tasks to be exactly the same ($t = 1$ and $t = 3$). Next, we use our method to learn skill embeddings and evaluate how many distinct embeddings it learns. The results in Fig. 2-right show the KL divergence between learned embedding distributions over training iterations. One can observe that the embedding network is able to discover that task 1 and 3 can be represented by the same skill embedding resulting in the KL-divergence between these embedding distribution being close to zero ($KL(p(z|t_1)||p(z|t_3)) \approx 0$)). This indicates that the embedding network is able to discover the number of distinct skills necessary to accomplish a set of tasks.

We present another didactic example demonstrating versatility of different solutions with the Gaussian embedding in the Appendix A.6.

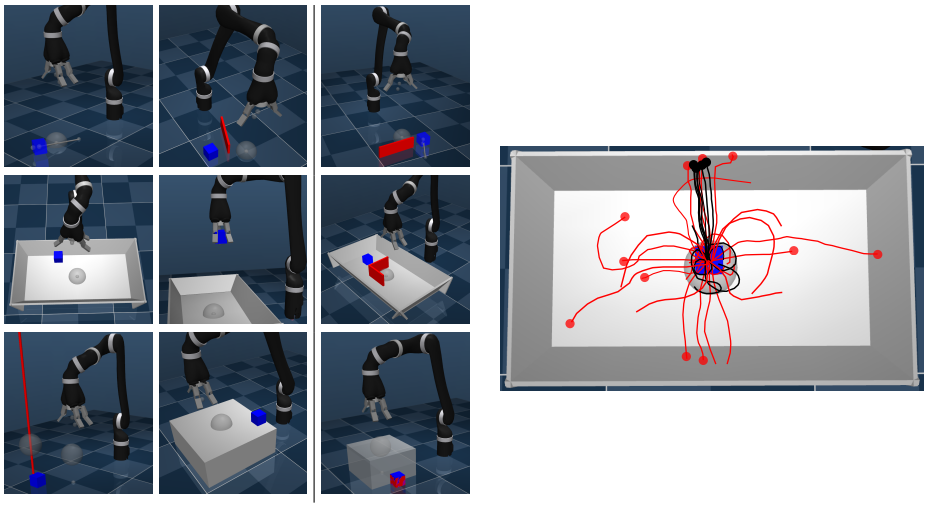

Figure 3: Left: visualization of the sequence of manipulation tasks we consider. Top row: spring-wall, middle row: L-wall, bottom row: rail-push. The left two columns depict the two initial skills that are learned jointly, the rightmost column depicts the transfer task that should be solved using the previously acquired skills. Right: Trajectories of the block in the plane as manipulated by the robot. The trajectories are produced by sampling a random embedding vector trained with (red) and without (black) the inference network from the marginal distribution over the L-wall pre-training tasks every 50 steps and following the policy. Dots denote points at which the block was lifted.

6.2 Control of the Skill Embedding for Manipulation Tasks

Next, we evaluate whether it is possible to use the learned skill embedding for control in an unseen scenario. We do so by using three simulated robotic manipulation tasks depicted in Fig. 3 and described below. The video of our experiments is available at: `https://goo.gl/FbvPGB`.

Spring-wall. A robotic arm is tasked to bring a block to a goal. The block is attached to a string that is attached to the ground at the initial block location. In addition, there is a short wall between the target and the initial location of the block, requiring the optimal behavior to pull the block around the wall and hold it at the goal location. The skill embedding used for learning this skill was learned on two tasks: bringing a block attached on a spring to a goal location (without a wall in between) and bringing a block to a goal location with a wall in between (without the spring). In order to successfully learn the new spring-wall skill, the skill-embedding space has to be able to *interpolate* between the skills it was originally trained on.

L-wall. The task is to bring a block to a goal that is surrounded by an L-shaped wall (see Fig. 3). The robot needs to learn how to push the block around the L-shaped wall to get to the target location. The skill embedding space used for learning this skill was learned on two tasks: push a block to a goal location (without the L-shaped wall) and lift a block to a certain height. The block was randomly spawned on a ring around the goal location that is in the center of the workspace.

Rail-push. The robot is tasked to first lift the block along the side of a white table that is firmly attached to the ground and then, to push it towards the center of the table. The initial lifting motion of the block is constrained as if the block was attached to a pole (or an upwards facing rail). This attachment is removed once the block reaches the height of the table. The skill embedding space was learned using two tasks: lift up the block attached on a rail (without the table in the scene) and push a block initialized on top of the table to its center.

The spring-wall and L-wall tasks are performed in a setting with sparse rewards (where the only reward the robot can obtain is tied to the box being inside a small region near a target location); making them very challenging exploration problems. In contrast, the rail-push task (due to its sequential nature as well as the fact that the table acts as an obstacle) uses minor reward shaping (where we additionally reward the robot based on the distance of the box to the center of the table).

Fig. 4 shows the comparison between our method and various baselines: i) learning the transfer task from scratch, ii) learning the mapping between states and the task id (t) directly without a stochastic skill-embedding space, iii) learning the task by controlling the skill-embedding that was trained without variational-inference-based regularization (no inference net).

In the spring-wall task, our approach has an advantage especially in the initial stages of training but the baseline without the inference network (no-KL in the plot) is able to achieve similar asymptotic performance. This indicates that this task does not require versatile skills and it is sufficient to find an embedding in between two skills that is able to successfully interpolate between them.

For the more challenging L-wall task, our method is considerably more successful than all the baselines. The agent has to discover an embedding that allows the robot to push the block along the edge of the white container - a behavior that is not directly required in any of the pre-training tasks. However, as it turns out, many successful policies for solving the lift task push the block against the wall of the container in order to perform a scooping motion. The agent is able to discover such a skill embedding and utilize it to push the block around the L-shaped wall.

In order to investigate why the baselines are not able to find a solution to the L-wall task, we explore the embedding space produced by our method as well as by the no-inference-network baseline. In particular, we sample a random embedding vector from the marginal embedding distribution over tasks and keep it constant to generate a behavior. The resulting trajectories of the block are visualized in Fig. 3-right. One can observe that the additional regularization causes the block trajectories to be much more versatile, which makes it easier to discover a working embedding for the L-wall task.

It is worth noting that the rail task used for initial training of the rail lift skill does not include the table. For the transfer task we, however, require the agent to find a skill embedding that is able to lift the block such that the arm is not in collision with the previously unseen table. As shown in the rightmost plot of Fig. 4, such an embedding is only discovered using our method. This indicates that due to the versatility of the learned skills, the agent is able to discover an embedding that avoids the collision with the previously unseen table and accomplishes the task successfully. The consecutive images of the final policies for all three tasks are presented in Appendix in Fig. 6.

Figure 4: Comparison of our method against different training strategies for our manipulation tasks: spring-wall, L-wall, and rail-push.

7 Conclusions

We presented a method that learns manipulation skills that are continuously parameterized in a skill embedding space, and takes advantage of these skills by solving a new control problem in the embedding space rather than the raw action space. Our experiments indicate that our method allows for discovery of multiple solutions and is capable of learning the minimum number of distinct skills that are necessary to solve a given set of tasks. In addition, we showed that our technique can interpolate and/or sequence previously learned skills in order to accomplish more complex tasks, even in the presence of sparse rewards.

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pp. 1726–1734, 2017.

[3] David Barber and Felix V. Agakov. The IM algorithm: A variational approach to information maximization. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pp. 201–208, 2003.

[4] Serkan Cabi, Sergio Gómez Colmenarejo, Matthew W Hoffman, Misha Denil, Ziyu Wang, and Nando de Freitas. The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. *arXiv preprint arXiv:1707.03300*, 2017.

[5] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. URL `http://arxiv.org/abs/1511.07289`.

[6] Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. *arXiv preprint arXiv:1706.06383*, 2017.

[7] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[9] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[10] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence UAI*, 2016.

[11] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[12] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

[13] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Neural Information Processing Systems (NIPS)*, 2017.

[14] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

[15] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[18] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, pp. 201611835, 2017.

[19] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, pp. 207–215, 2013.

[20] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[21] Yunzhu Li, Jiaming Song, and Stefano Ermon. Inferring the latent structure of human decision-making from raw visual inputs. *arXiv preprint arXiv:1703.08840*, 2017.

[22] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

[23] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations (ICLR)*, 2017.

[24] Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *CoRR*, abs/1707.02201, 2017.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

[27] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pp. 2125–2133, 2015.

[28] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 1046–1054, 2016. URL `http://papers.nips.cc/paper/6538-safe-and-efficient-off-policy-reinforcement-learning`.

[29] Gerhard Neumann. Variational inference for policy search in changing situations. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 817–824, 2011.

[30] Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International Conference on Machine Learning*, pp. 324–333, 2016.

[31] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *(R:SS 2012)*, 2012. *Runner Up Best Paper Award*.

[32] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

[33] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[34] T. Salimans, D. P. Kingma, and M. Welling. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. *ArXiv e-prints*, October 2014.

[35] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015.

[36] John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[37] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

[38] Emanuel Todorov. General duality between optimal control and estimation. In *Proceedings of the 47th IEEE Conference on Decision and Control, CDC 2008, December 9-11, 2008, Cancún, México*, pp. 4286–4292, 2008.

[39] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 1049–1056, 2009. ISBN 978-1-60558-516-1.

[40] Ziyu Wang, Josh Merel, Scott E. Reed, Greg Wayne, Nando de Freitas, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, 2017.

[41] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[42] Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Machine Learning Department, Carnegie Mellon University, Dec 2010.

A Appendix

A.1 Preliminaries

We perform reinforcement learning in Markov decision processes (MDP). We denote with $s \in \mathbb{R}^S$ the continuous state of the agent; $a \in \mathbb{R}^A$ denotes the action vector and $p(s_{t+1}|s_t, a_t)$ the probability of transitioning to state s_{t+1} when executing action a_t in s_t. Actions are drawn from a policy distribution $\pi_\theta(a|s)$, with parameters θ; in our case a Gaussian distribution whose mean and diagonal covariance are parameterized via a neural network. At every step the agent receives a scalar reward $r(s_t, a_t)$ and we consider the problem of maximizing the sum of discounted rewards $\mathbb{E}_{\tau_\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, s_t)]$.

A.2 Off-policy Learning Details

As described in the main paper we make use of the recent Retrace algorithm from [28], which allows us to quickly propagate entropy augmented rewards across multiple time-steps while – at the same time – minimizing the bias that any algorithm relying on a parametric Q-function is prone to. Formally, we fit Q^π_φ by minimizing the squared loss:

$$\begin{aligned}
&\min_{\varphi} \mathbb{E}_{\mathcal{B}}\Big[\big(Q^\pi_\varphi(s_i, a_i; z, t) - Q^{\text{ret}}\big)^2\Big], \text{with} \\
&Q^{\text{ret}} = \sum_{j=i}^{\infty} \Big(\gamma^{j-i} \prod_{k=i}^{j} c_k\Big)\Big[\hat{r}(s_j, a_j, z, t) + \mathbb{E}_{\pi(a|z,s,t)}[Q^b_{\varphi'}(s_i, \cdot; z, t)] - Q^b_{\varphi'}(s_j, a_j; z, t)\Big], \qquad (7) \\
&c_k = \min\Big(1, \frac{\pi(a_k|z, s_k, t)p(z|t)}{b(a_k|z, s_k, t)b(z|t)}\Big),
\end{aligned}$$

where we compute the terms contained in $\hat{r}$ by using r_t and z from the replay buffer and re-compute the (cross-)entropy terms. Here, φ' denotes the parameters of a target Q-network[3] [25] that we occasionally copy from the current estimate φ, and c_k are the per-step importance weights. Further, we bootstrap the infinite sum after N-steps with $E_\pi\Big[Q^\pi_{\varphi'}(s_N, \cdot; z_N, t)\Big]$ instead of introducing a λ parameter as in the original paper [28].

A.3 Variational Bound Derivation

In order to introduce an information-theoretic regularization that encourages versatile skills, we borrow ideas from the variational inference literature. In particular, in the following, we present a lower bound of the marginal entropy $\mathcal{H}(p(x))$, which will prove useful when applied to the reinforcement learning objective in Sec. 3.1.

Theorem 1. *The lower bound on the marginal entropy $\mathcal{H}(p(x))$ corresponds to:*

$$\mathcal{H}(p(x)) \geq \int\int p(x, z) \log(\frac{q(z|x)}{p(x, z)} dz) dx, \qquad (8)$$

where $q(z|x)$ is the variational posterior.

Proof.

$$\begin{aligned}
\mathcal{H}(p(x)) &= \int p(x) \log(\frac{1}{p(x)}) dx = \int p(x) \log(\int q(z|x) \frac{1}{p(x)} dz) dx \\
&= \int p(x) \log(\int q(z|x) \frac{p(z|x)}{p(x, z)} dz) dx \geq \int p(x) \int p(z|x) \log(\frac{q(z|x)}{p(x, z)} dz) dx \\
&= \int\int p(x, z) \log(\frac{q(z|x)}{p(x, z)} dz) dx. \qquad (9)
\end{aligned}$$

□

[3] Note it will thus evaluate a different policy than the current policy π, here denoted by b. Nonetheless by using importance weighting via c_k we are guaranteed to obtain an unbiased estimator in the limit.

A.4 Derivation for Multiple Timesteps

We represent the trajectory as $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ and the learned parametrized posterior (policy) as $\pi_\theta(\tau) = p(s_0)\prod_{i=0}^{T-1}\pi_\theta(a_i|s_i)p(s_{i+1}|s_i, a_i)$. The learned inference network is represented by $q_\phi(z|\tau)$ and we introduce the pseudo likelihood that is equal to cumulative reward: $\log p(R = 1|\tau) = \sum_t r(s_t, a_t)$.

In this derivation we also assume the existence of a prior over trajectories of the form: $\mu(\tau) = p(s_0)\prod_{i=0}^{T-1}\mu(a_i|s_i)p(s_{i+1}|s_i, a_i)$. where μ represents our "prior policy". We thus consider the relative entropy between π and μ. Note that we can choose prior policy to be non-informative (e.g. a uniform prior over action for bounded action spaces).

With these definitions, we can cast RL as a variational inference problem:

$$\begin{aligned}
L = \log \int p(R=1|\tau)\mu(\tau)d\tau &\geq \int \pi(\tau)\log\frac{p(R=1|\tau)\mu(\tau)}{\pi(\tau)}d\tau \\
&= \int \pi(\tau)\log p(R=1|\tau)d\tau + \int \pi(\tau)\log\frac{\mu(\tau)}{\pi(\tau)}d\tau \\
&= \mathbb{E}_\pi[\sum_t r(s_t, a_t)] + \mathbb{E}_\pi\left[\sum_t \log\frac{\mu(a_t|s_t)}{\pi(a_t|s_t)}\right] \\
&= \mathbb{E}_\pi[\sum_t r(s_t, a_t)] + \mathbb{E}_\pi\left[\sum_t \mathrm{KL}[\pi_t||\mu_t]\right] = \mathcal{L}, \qquad (10)
\end{aligned}$$

We can now introduce the latent variable z that forms a Markov chain:

$$\begin{aligned}
\pi(\tau) &= \int \pi(\tau|z)p(z)dz \\
&= \int p(s_0)p(z_0)\prod_{i=0}^{T-1}\pi(a_i|s_i, z_i)p(s_{i+1}|s_i, a_i)p(z_{i+1}|z_i)dz_{1:T}. \qquad (11)
\end{aligned}$$

Applying it to the loss, we obtain:

$$\begin{aligned}
\mathcal{L} &= \mathbb{E}_\pi[\sum_t r(s_t, a_t)] + \mathbb{E}_\pi[\mathrm{KL}[\pi(\tau)||\mu(\tau)]] \\
&= \mathbb{E}_\pi[\sum_t r(s_t, a_t)] + \mathbb{E}_\pi\left[\log\frac{\mu(\tau)}{\int \pi(\tau|z_{1:T})p(z_{1:T})dz_{1:T}}d\tau\right] \\
&\geq \mathbb{E}_\pi[\sum_t r(s_t, a_t)] + \mathop{\mathbb{E}}_{\pi(\tau)}\mathop{\mathbb{E}}_{p(z_{1:T}|\tau)}\left[\sum_t \int \pi(a'_t|s_t, z_t)\log\frac{\mu(a'_t|s_t)}{\pi(a'_t|s_t, z_t)}da'_t + \log\frac{q(z_{1:T}|\tau)}{p(z_{1:T})}\right]. \qquad (12)
\end{aligned}$$

Equation (12) arrives at essentially the same bound as that in Equation (2) but for sequences. The exact form of (12) in the previous equation depends on the form that is chosen for q. For instance, for $q(z|\tau) = q(z_T|\tau)q(z_{T-1}|z_T, \tau)q(z_{T-2}|z_{T-1}, \tau)\ldots$ we obtain:

$$\begin{aligned}
&\mathbb{E}_\pi\left[\sum_t \log\frac{\mu(a_t|s_t)}{\pi(a_t|s_t, z_t)} + \log\frac{q(z_{1:T}|\tau)}{p(z_{1:T})}\right] \\
=&\mathbb{E}_\pi\left[\sum_t \log\frac{\mu(a_t|s_t)}{\pi(a_t|s_t, z_t)} + \sum_{t=1}^{T}\log\frac{q(z_{t-1}|z_t, \tau)}{p(z_{t+1}|z_t)} + \log q(z_T|\tau) - \log p(z_0)\right]. \qquad (13)
\end{aligned}$$

Other forms for q are also feasible, but the above form gives a nice temporal decomposition of the (augmented) reward.

A.5 Algorithm details

A.5.1 Stochastic Value Gradient for the policy

We here give a derivation of the stochastic value gradient for the objective from Equation (5) that we use for gradient based optimization. We start by reparameterizing the sampling step $z \sim p_\phi(z|t)$ for the embedding as $g_\phi(t, \epsilon_z)$, where ϵ_z is a random variable drawn from an appropriately chosen base distribution. That is, for a Gaussian embedding we can use a normal distribution [17, 32] $\epsilon_z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where I denotes the identity. For a Bernoulli embedding we can use the Concrete distribution reparametrization [23] (also named the Gumbel-softmax trick [16]). For the policy distribution we always assume a Gaussian and can hence reparameterize using $g_\theta(t, \epsilon_a)$ with $\epsilon_a \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Using a Gaussian embedding we then get the following gradient for the the policy parameters θ

$$\begin{aligned}\nabla_\theta \hat{L}(\theta, \phi) &= \nabla_\theta \left[\mathbb{E}_{\substack{\pi_\theta(a|z,s) \\ p_\phi(z|t) \\ s,t \in \mathcal{B}}} \Big[Q^\pi_\varphi(s, a, z) \Big] + \mathbb{E}_{t \in \mathcal{T}} \Big[\mathcal{H}[p_\phi(z|t)] \Big] \right], \\ &= \mathbb{E}_{\substack{\epsilon_a \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \\ \epsilon_z \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \\ s,t \in \mathcal{B}}} \Big[\nabla_\theta \, Q^\pi_\varphi(s, g_\theta(t, \epsilon_a), g_\phi(t, \epsilon_z)) \nabla_\theta g_\theta(t, \epsilon_a) \Big],\end{aligned} \tag{14}$$

and, for the embedding network parameters,

$$\begin{aligned}\nabla_\phi \hat{L}(\theta, \phi) &= \nabla_\phi \left[\mathbb{E}_{\substack{\pi_\theta(a|z,s) \\ p_\phi(z|t) \\ s,t \in \mathcal{B}}} \Big[Q^\pi_\varphi(s, a, z) \Big] + \mathbb{E}_{t \in \mathcal{T}} \Big[\mathcal{H}[p_\phi(z|t)] \Big] \right], \\ &= \mathbb{E}_{\substack{\epsilon_a \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \\ \epsilon_z \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \\ s,t \in \mathcal{B}}} \Big[\nabla_\phi \, Q^\pi_\varphi(s, g_\theta(t, \epsilon_a), g_\phi(t, \epsilon_z)) \nabla_\phi g_\phi(t, \epsilon_z) \Big] + \mathbb{E}_{t \in \mathcal{T}} \Big[\nabla_\phi \mathcal{H}[p_\phi(z|t)] \Big].\end{aligned} \tag{15}$$

A.6 Another Didactic Example

The second didactic example consists of a force-controlled point mass that is rewarded for being in a goal region. In order to learn the skill embedding, we use two tasks ($T = 2$), with the goals located either to the left or to the right of the initial location.

Fig. 5-bottom compares a set of trajectories produced by our method when conditioned on different Gaussian skill embedding samples with and without the variational-inference-based regularization. The hereby introduced cross-entropy term between inference and embedding distributions introduces more variety to the obtained trajectories, which can be explained by the agent's incentive to help the inference network. Fig. 5-top presents the absolute error between the actual and the inferred skill embedding for both tasks. It is apparent that the trajectories generated with regularization, display more variability and are therefore easily distinguishable. The constant residual error shown in the top left part of the figure corresponds to the fact that the inference network without regularization can only predict the mean of the embedding used for generating the trajectories.

A.7 Implementation Details

A.7.1 Task Structure

All the tasks presented in Sec. 6.2 share a similar structure, in that the observation space used for the pre-trained skills and the observation space used for the final task are the same. For all three tasks, the observations include: joint angles (6) and velocities (6) of the robot joints, finger joints positions (3) and velocities (3), position of the endeffector (3), position (3), orientation (4) and linear velocity (3) of the block as well as the position of the goal (3). The action space is also the same across all tasks and consists of joint torques for all the robot joints including the hand (9). We choose such a structure (making sure that the action space matches and providing only proprioceptive information to the policy) to make sure we i) can transfer the policy between tasks directly; 2) to ensure that the only way the agent is informed about changing environment dynamics (e.g., the attachment of the block to a string, the existence of a wall, etc.) is through the task id.

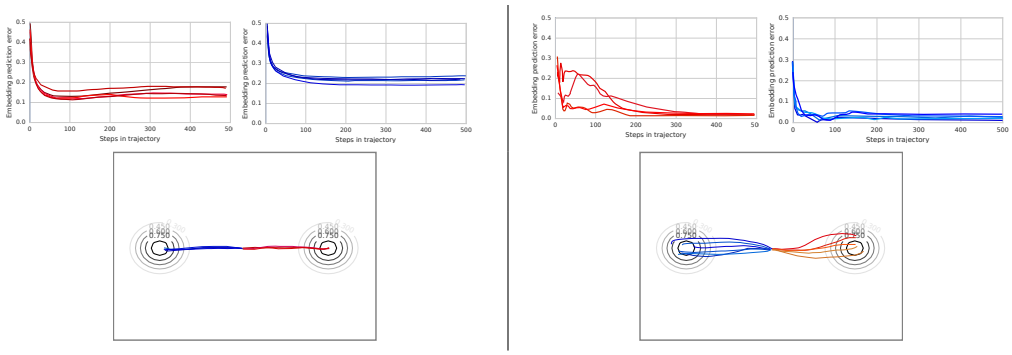

Figure 5: Bottom: resulting trajectories for different 3D embedding values with (right) and without (left) variational-inference-based regularization. The contours depict the reward gained by the agent. Top: Absolute error between the mean embedding value predicted by the inference network and the actual mean of the embedding used to generate these trajectories. Note that every error curve at the top corresponds to a single trajectory at the bottom.

The rationale behind having the same observation space between the pre-trained skills and the final task comes from the fact that currently, our architecture expects the same observations for the final policy over embeddings and the skill subpolicies. We plan to address this limitation in future work.

A.7.2 Network Architecture and Hyperparameters

The hereby presented values were used to generate results for the final three manipulation tasks presented in Sec. 6.2. For both policy and inference network we used two-layer fully connected neural networks with exponentiaded linear activations [5] (for layer sizes see table) to parameterize the distribution parameters. As distributions we always relied on a gaussian distribution $\mathcal{N}(\mu_\theta(x), \text{diag}(\sigma(x)))$ whose mean and diagonal covariance are parameterized by the policy network via $[mu(x), \log \sigma(x)] = f_\theta(x)$. For the embedding network the mapping from one-hot task vectors to distribution parameters is given via a linear transformation. For the inference network we map to the parameters of the same distribution class via another neural network.

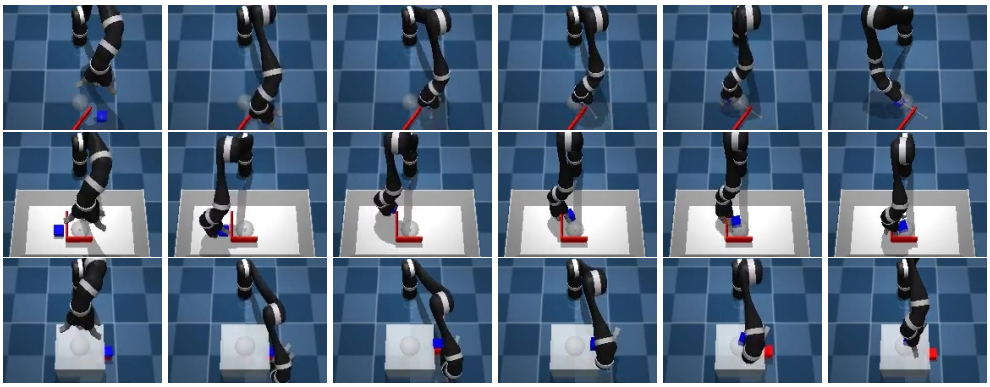

Figure 6: Final policy for all three tasks: spring-wall (top), L-wall (middle), rail-push (bottom).

Hyperparameter	Spring-wall	L-wall	Rail-push
State dims	34	34	34
Action dims	9	9	9
Policy net	100-100	100-100	100-100
Q function net	200-200	200-200	200-200
Inference net	100-100	100-100	100-100
Embedding distribution	3D Gaussian	3D Gaussian	3D Gaussian
Minibatch size (per-worker)	32	32	32
Replay buffer size	$1e^5$	$1e^5$	$1e^5$
α_1	10^{-4}	10^{-4}	10^{-4}
α_2	10^{-5}	10^{-5}	10^{-5}
α_3	10^{-4}	10^{-4}	10^{-4}
Discount factor (γ)	0.99	0.99	0.99
Adam learning rate	10^{-4}	10^{-4}	10^{-4}

Table 1: Hyperparameters used in the experiments.