Regularized Hierarchical Policies for Compositional Transfer in Robotics

Markus Wulfmeier*, Abbas Abdolmaleki*, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, Martin Riedmiller

DeepMind
London, United Kingdom

Abstract

The successful application of flexible, general learning algorithms — such as deep reinforcement learning — to real-world robotics applications is often limited by their poor data-efficiency. Domains with more than a single dominant task of interest encourage algorithms that share partial solutions across tasks to limit the required experiment time. We develop and investigate simple hierarchical inductive biases — in the form of structured policies — as a mechanism for knowledge transfer across tasks in reinforcement learning (RL). To leverage the power of these structured policies we design an RL algorithm that enables stable and fast learning. We demonstrate the success of our method both in simulated robot environments (using locomotion and manipulation domains) as well as real robot experiments, demonstrating substantially better data-efficiency than competitive baselines.

1 Introduction

While recent successes in deep (reinforcement) learning for computer games (Atari [Mnih et al., 2013], StarCraft [Vinyals et al., 2019], Go [Silver et al., 2017] and other high-throughput domains, e.g. [OpenAI et al., 2018], have demonstrated the potential of these methods in the big data regime, high costs of data acquisition limit progress in many tasks of real-world relevance. Data efficiency in machine learning often relies on inductive biases to guide and accelerate the learning process; e.g. by including expert domain knowledge of varying granularity. While incorporating this knowledge often helps to accelerate learning, when inaccurate it can inappropriately bias the space of solutions and lead to sub-optimal results.

Instead of designing inductive biases manually we aim to infer them from related tasks. In particular, we turn towards research on multitask transfer learning, where, from the view of one task, training signals of related tasks can serve as an additional inductive bias [Caruana, 1997]. Focusing on domains with consistent algorithm embodiment and more than a single task of interest – such as robotics – sharing and applying knowledge across tasks is imperative for efficiency and scalability. Successes for transfer learning have, for example, built on optimizing initial parameters [e.g. Finn et al., 2017], sharing models and parameters across tasks [e.g. Rusu et al., 2016, Teh et al., 2017, Galashov et al., 2018], data-sharing across tasks [e.g. Riedmiller et al., 2018, Andrychowicz et al., 2017] or related, auxiliary objectives [Jaderberg et al., 2016, Wulfmeier et al., 2017]. Transfer between tasks can, however, lead to either constructive or destructive transfer for humans [Singley and Anderson, 1989] as well as machines [Pan and Yang, 2010, Torrey and Shavlik, 2010]. That is, jointly learning to solve different tasks can provide benefits as well as disadvantages for individual tasks, depending on the similarity of required solutions.

Preprint. Under review.

*(Shared first-authorship. Correspondence to: mwulfmeier,aabdolmaleki@google.com)

In reinforcement learning (RL), (partial) solutions can be shared e.g. in the form of policies or value functions. While training independent, per-task, policies prevents interference, it also prevents transfer. On the other hand, training a monolithic policy shared across tasks can instead cause interference of opposing modes of behaviour. With this work we continue to explore the benefits of hierarchy for multitask learning and the modelling of multi-modal distributions [Rosenstein et al., 2005, Bishop, 1994].

We focus on developing an actor-critic RL approach for training hierarchical, modular models (policies) for continuous control. We represent low-level sub-policies as Gaussian policies combined with a categorical distribution for the high-level controller. We parametrize the policies as simple Mixture Density Networks [Bishop, 1994] with the important modification of providing task information only to the high-level controller, so that the low-level policies acquire more robust, task-independent behaviours. To train hierarchical policies in a coordinated and robust manner, we extend the optimization scheme of the Maximum a-posteriori Policy Optimization algorithm (MPO) [Abdolmaleki et al., 2018a]. We additionally demonstrate the general benefits of hierarchical policies for transfer learning via improved results for another, gradient-based, entropy regularized RL algorithm [Heess et al., 2016] (see Appendix A.8.4).

The resulting algorithm, Regularized Hierarchical Policy Optimization (RHPO), is appealing due to its conceptual simplicity and appears extremely effective in practice. We demonstrate this first in simulation for a variety of continuous control problems. In multitask transfer problems, we find that RHPO can improve learning speed compared to the state-of-the-art. We then compare the approach against competitive continuous control baselines on a real robot for robotic manipulation tasks where RHPO leads to a significant speed up in training, enabling us to learn with increased efficiency in a challenging stacking domain on a single robot. To the best of our knowledge, RHPO demonstrates the first success for learning such a challenging task, defined via a sparse reward, from raw robot state and a randomly initialized policy in the real world – which we can learn within about a week of real robot time[2].

2 Preliminaries

We consider a multitask reinforcement learning setting with an agent operating in a Markov Decision Process (MDP) consisting of the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition probability $p(s_{t+1}|s_t, a_t)$ of reaching state s_{t+1} from state s_t when executing action a_t at the previous time step t. The actions are drawn from a probability distribution over actions $\pi(a|s)$ referred to as the agent's policy. Jointly, the transition dynamics and policy induce the marginal state visitation distribution $p(s)$. Finally, the discount factor γ together with the reward $r(s, a)$ gives rise to the expected reward, or value, of starting in state s (and following π thereafter) $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^\infty \gamma^t r(s_t, a_t)|s_0 = s, a_t \sim \pi(\cdot|s_t, i), s_{t+1} \sim p(\cdot|s_t, a_t)]$. Furthermore, we define multitask learning over a set of tasks $i \in I$ with common agent embodiment as follows. We assume shared state, action spaces and shared transition dynamics across tasks; tasks only differ in their reward function $r_i(s, a)$. Furthermore, we consider task conditional policies $\pi(a|s, i)$. The overall objective is defined as

$$J(\pi) = \mathbb{E}_{i\sim I}\left[\mathbb{E}_{\pi,p(s_0)}\left[\sum_{t=0}^{\infty}\gamma^t r_i\left(s_t, a_t\right)|s_{t+1}\sim p(\cdot|s_t, a_t)\right]\right] = \mathbb{E}_{i\sim I}\Big[\mathbb{E}_{\pi,p(s)}\big[Q^\pi(s, a, i)\big]\Big], \tag{1}$$

where all actions are drawn according to the policy π, that is, $a_t \sim \pi(\cdot|s_t, i)$ and we used the common definition of the state-action value function – here conditioned on the task – $Q^\pi(s, a, i) = \mathbb{E}_\pi\left[\sum_{t=0}^\infty \gamma^t r_i\left(s_t, a_t\right)|a_0 = a, s_0 = s, a_t \sim \pi(\cdot|s_t, i), s_{t+1} \sim p(\cdot|s_t, a_t)\right]$.

3 Method

This section introduces Regularized Hierarchical Policy Optimization (RHPO) which focuses on efficient training on modular policies by sharing data across tasks building on an instance of Scheduled Auxiliary Control (SAC-X) with randomized scheduling (SAC-U) [Riedmiller et al., 2018]. We start by introducing the considered class of policies, followed by the required extension of the MPO

[2] Additional videos are provided under `https://sites.google.com/view/rhpo/`

algorithm [Abdolmaleki et al., 2018b,a] for training structured hierarchical policies in an off-policy setting.

3.1 Modular Hierarchical Policies

We start by defining the modular policy class which supports sharing sub-policies across tasks. Formally, we decompose the per-task policy $\pi(a|s,i)$ as

$$\pi_\theta(a|s,i) = \sum_{o=1}^{M} \pi_\theta^L\left(a|s,o\right)\pi_\theta^H\left(o|s,i\right), \tag{2}$$

with π^H and π^L respectively representing a "high-level" switching controller (a categorical distribution) and a "low-level" sub-policy and o is the index of the sub-policy. Here, θ denotes the parameters of both π^H and π^L, which we will seek to optimize. While the number of components has to be decided externally, the method is robust with respect to this parameter (Appendix A.8.1).

Note that, in the above formulation only the high-level controller π_H is conditioned on the task information i; i.e. we employ a form of information asymmetry [Galashov et al., 2018, Tirumala et al., 2019, Heess et al., 2016] to enable the low-level policies to acquire general, task-independent behaviours. This choice strengthens decomposition of tasks across domains and inhibits degenerate cases of bypassing the high-level controller. These sub-policies can be understood as building reflex-like low-level control loops, which perform domain-dependent useful behaviours and can be modulated by higher cognitive functions with knowledge of the task at hand.

3.2 Decoupled Hierarchical Multitask Optimization

In the following sections, we present the equations underlying RHPO, for the complete pseudocode algorithm the reader is referred to the Appendix A.2.1. To optimize the policy class described above we build on the MPO algorithm [Abdolmaleki et al., 2018b,a] which decouples the policy improvement step (optimizing J independently of the policy structure) from the fitting of the hierarchical policy. Concretely, we first introduce an intermediate non-parametric policy $q(a|s,i)$ and consider optimizing $J(q)$ while staying close, in expectation, to a reference policy $\pi_{ref}(a|s,i)$

$$\max_q J(q,\pi_{ref}) = \mathbb{E}_{i\sim I}\Big[\mathbb{E}_{\pi,s\sim\mathcal{D}}\big[\hat{Q}(s,a,i)\big]\Big], \text{s.t. } \mathbb{E}_{s\sim\mathcal{D},i\sim I}\quad\Big[\mathrm{KL}\big(q(\cdot|s,i)\|\pi_{ref}(\cdot|s,i)\big)\Big] \leq \epsilon, \tag{3}$$

where $\mathrm{KL}(\cdot\|\cdot)$ denotes the Kullback Leibler divergence, ϵ defines a bound on the KL, $\mathcal{D}$ denotes the data contained in a replay buffer (note that we do not differentiate between states from different tasks here, see Section 3.3 for further explanation), and assuming that we have an approximation of the ground-truth state-action value function $\hat{Q}(s,a,i) \approx Q^\pi(s,a,i)$ available (see Section 3.3 for details on how $\hat{Q}$ can be learned from off-policy data). Starting from an initial policy π_{θ_0} we can then iterate the following steps (interleaved with policy evaluation to estimate Q) to improve the policy π_{θ_k}:

- **Policy Evaluation:** Update $\hat{Q}$ such that $\hat{Q}(s,a,i) \approx \hat{Q}^{\pi_{\theta_k}}(s,a,i)$, see Section 3.3.
- **Step 1:** Obtain $q_k = \arg\max_q J(q,\pi_{\theta_k})$, under KL constraints (Equation (3)).
- **Step 2:** Obtain $\theta_{k+1} = \arg\min_\theta \mathbb{E}_{s\sim\mathcal{D},i\sim I}\Big[\mathrm{KL}\big(q_k(\cdot|s,i)\|\pi_\theta(\cdot|s,i)\big)\Big]$, under additional regularization (Equation (5)).

Step 1: Obtaining Non-parametric Policies We first find the intermediate policy q by maximizing Equation (3). Analogous to MPO [Abdolmaleki et al., 2018b,a], we obtain a closed-form solution with a non-parametric policy for each task, as

$$q_k(a|s,i) \propto \pi_{\theta_k}(a|s,i)\exp\left(\frac{\hat{Q}(s,a,i)}{\eta}\right), \tag{4}$$

where η is a temperature parameter (corresponding to a given bound ϵ) which is optimized alongside the policy optimization. See Abdolmaleki et al. [2018b,a] for the general form as well as the Appendix A.1.1 for a detailed derivation for the multi-task case. As mentioned above, this policy representation

is independent of the form of the parametric policy π_{θ_k}; i.e. q only depends on π_{θ_k} through its density. This, crucially, makes it easy to employ complicated structured policies (such as the one introduced in Section 3.1). The only requirement for this and the following steps is that we must be able to sample from π_{θ_k} and calculate the gradient (w.r.t. θ_k) of its log density.

Step 2: Fitting Parametric Policies In the second step we fit a policy to the non-parametric distribution obtained from the previous calculation by minimizing the divergence $\mathbb{E}_{s\sim\mathcal{D},i\sim I}[\mathrm{KL}(q_k(\cdot|s,i)\|\pi_\theta(\cdot|s,i))]$. Assuming that we can sample from q_k this step corresponds to maximum likelihood estimation (MLE). Furthermore, we can effectively mitigate optimization instabilities during training by placing a prior on the change of the policy (a trust-region constraint). The idea is commonly used in on- as well as off-policy RL [Schulman et al., 2015, Abdolmaleki et al., 2018b,a]. The application to hierarchical policy classes highlights the importance of this constraint as investigated in Section 4.1. Formally, we aim to obtain the solution

$$\begin{aligned}\theta_{k+1} &= \arg\min_\theta \mathbb{E}_{s\sim\mathcal{D},i\sim I}\Big[\mathrm{KL}\big(q_k(\cdot|s,i)\|\pi_\theta(\cdot|s,i)\big)\Big] \\ &= \arg\max_\theta \mathbb{E}_{s\sim\mathcal{D},i\sim I}\left[\mathbb{E}_{\pi_k}\Big[\exp(\hat{Q}(s,a,i)/\eta)\log\sum_{o=1}^{M}\pi_\theta^L(a|s,o)\,\pi_\theta^H(o|s,i)\Big]\right], \\ \text{s.t. } &\mathbb{E}_{s\sim\mathcal{D},i\sim I}\left[\mathrm{KL}(\pi_{\theta_k}^H(o|s,i)\|\pi_\theta^H(o|s,i)) + \frac{1}{M}\sum_{o=1}^{M}\mathrm{KL}(\pi_{\theta_k}^L(a|s,o)\|\pi_\theta^L(a|s,o))\right] < \epsilon_m,\end{aligned} \tag{5}$$

where ϵ_m defines a bound on the change of the new policy. Here we drop constant terms and the negative sign in the second line (turning $\min$ into $\max$), and explicitly insert the definition $\pi_\theta(a|s,i) = \sum_{o=1}^{M}\pi_L(a|s,o)\,\pi_H(o|s,i)$, to highlight that we are marginalizing over the high-level choices in this fitting step (since q is not tied to the policy structure). Hence, the update is independent of the specific policy component from which the action was sampled, enabling joint updates of all components. This reduces the variance of the update and also enables efficient off-policy learning. Different approaches can be used control convergence for both the "high-level" categorical choices and the action choices to change slowly throughout learning. The average KL constraint in Equation (5) is similar in nature to a upper bound on the computationally intractable KL divergence between the two mixture distributions and has been determined experimentally to perform better in practice than simple bounds. In practice, inspired by Abdolmaleki et al. [2018a], in order to control the change of the high level and low level policies independently we decouple the constraints to be able to set different ϵ for the means (ϵ_μ), covariances (ϵ_Σ) and the categorical distribution (ϵ_α) in case of a mixture of Gaussian policy. Please see the Appendix A.1.2 for more details on decoupled constraints. To solve Equation (5), we first employ Lagrangian relaxation to make it amenable to gradient based optimization and then perform a fixed number of gradient descent steps (using Adam [Kingma and Ba, 2014]); details on this step, as well as an algorithm listing, can be found in the Appendix A.1.2.

3.3 Data-efficient Multitask Learning

For data-efficient off-policy learning of $\hat{Q}$ we consider the same setting as scheduled auxiliary control setting (SAC-X) [Riedmiller et al., 2018] which introduces two main ideas to obtain data-efficiency: i) experience sharing across tasks; ii) switching between tasks within one episode for improved exploration. We consider uniform random switches between tasks every N steps within an episode (the SAC-U setting from [Riedmiller et al., 2018]).

Formally, we assume access to a replay buffer containing data gathered from all tasks, which is filled asynchronously to the optimization (similar to e.g. Espeholt et al. [2018], Abdolmaleki et al. [2018a]) where for each trajectory snippet $\tau = \{(s_0, a_0, R_0), \ldots, (s_L, a_L, R_L)\}$ we record the rewards for all tasks $R_t = [r_{i_1}(s_t, a_t), \ldots, r_{i_{|I|}}(s_t, a_t)]$ as a vector in the buffer. Using this data we define the retrace objective for learning $\hat{Q}$, parameterized via ϕ, following Riedmiller et al. [2018] as

$$\min_\phi L(\phi) = \sum_{i\sim I}\mathbb{E}_{\tau\sim\mathcal{D}}\Big[\big(r_i(s_t,a_t) + \gamma Q^{ret}(s_{t+1},a_{t+1},i) - \hat{Q}_\phi(s_t,a_t,i)\big)^2\Big], \tag{6}$$

where Q^{ret} is the L-step retrace target [Munos et al., 2016], see the Appendix A.2.2 for details.

4 Experiments

In the following sections, we investigate how RHPO can provide compelling benefits for multitask transfer learning in real and simulated robotic manipulation tasks. This includes experiments on physical hardware with robotic manipulation tasks for the Sawyer arm, emphasizing the importance of data-efficiency. In addition, we provide results for single-task domains from the DeepMind Control Suite [Tassa et al., 2018] in Appendix A.5.4, where we demonstrate how hierarchical policies can be employed to flexibly integrate human domain knowledge.

All policies are trained via the same reward structures, underlying constraints and task hyperparameters. More details on these parameters and the complete results for additional ablations and all tasks from the multitask domains are provided in the Appendix A.4. Across all tasks, we build on a distributed actor-critic framework (similar to [Espeholt et al., 2018]) with flexible hardware assignment [Buchlovsky et al., 2019] to train all agents, performing critic and policy updates from a replay buffer, which is asynchronously filled by a set of actors.

4.1 Simulated Robot Experiments

We use three simulated multitask scenarios with the Kinova Jaco and Rethink Robotics Sawyer robot arms to test in a variety of conditions. **Pile1**: Here, the seven tasks of interest range from simple reaching for a block over tasks like grasping it, to the final task of stacking the block on top of another block. In addition to the experiments in simulation, which are executed with 5 actors in a distributed setting, the same **Pile1** multitask domain (same rewards and setup) is investigated with a single, physical robot in Section 4.2. In all figures with error bars, we visualize mean and variance derived from 3 runs. We further extend the evaluation towards two more complex multitask domains in simulation. The first extension includes stacking with both blocks on top of the respective other block, resulting in a setting with 10 tasks (**Pile2**). And a last domain including harder tasks such as opening a box and placing blocks into this box, consisting of a total of 13 tasks (**Cleanup2**).

Across all experiments, we utilize the data-sharing scheme of Scheduled Auxiliary Control (SAC-X) [Riedmiller et al., 2018] (see Section 3.3), and execute a series of random tasks per episode, i.e. our setup is comparable to SAC-U. We thus compare RHPO for training **hierarchical** policies against a flat, monolithic policy shared across all tasks which is provided with the additional task id as input (displayed as **Monolithic** in the plot) as well as policies with task dependent heads (displayed as **Independent** in the plots) following [Riedmiller et al., 2018] – both using MPO as the optimizer and a re-implementation of SAC-U using SVG [Heess et al., 2015]. The baselines aim to provide the two opposite, naive perspectives on transfer: by using the same monolithic policy across tasks we enable positive as well as negative interference and independent policies prevent policy-based transfer. After experimentally confirming the robustness of RHPO with respect to the number of low-level sub-policies (see Appendix A.8.1), we set M equal to the number of tasks in each domain.

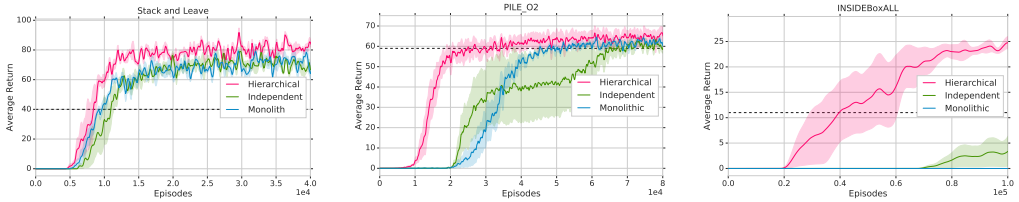


Figure 1: Results for the multitask robotic manipulation experiments in simulation. The dashed line corresponds to the performance of a re-implementation of SAC-U. From left to right: Pile1, Pile2, Cleanup2. We show averages over 3 runs each, with corresponding standard deviation. RHPO outperforms both baselines across all tasks with the benefits increasing for more complex tasks.

Figure 1 demonstrates that the hierarchical policy (RHPO) outperforms the monolithic as well as the independent baselines. For simple tasks such as the stacking of a one block on another (Pile1), the difference is comparably small, but the more tasks are trained and the more complex the domain becomes (cf. Pile2 and Cleanup2), the greater is the advantage of sharing learned behaviours across tasks. Compared to a re-implementation of SAC-U (using SVG as the optimizer) – dashed line in

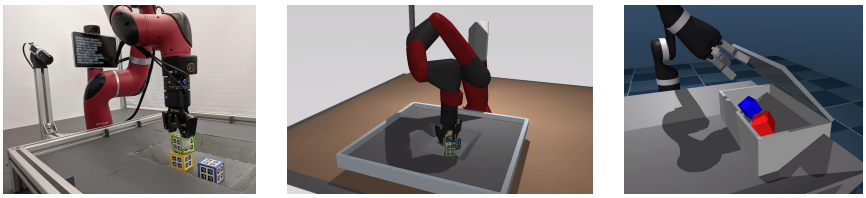


Figure 2: Left: Overview of the real robot setup with the Sawyer robot performing the Pile1 task. Screen pixelated for anonymization. Middle: Simulated Sawyer performing the same task. Right: Cleanup2 setup with the Jaco.

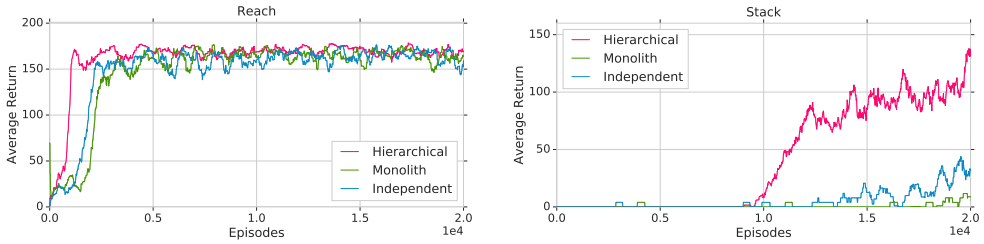
the plots – we observe that the baselines based on MPO already result in an improvement, which becomes even bigger with the hierarchical policies.

4.2 Physical Robot Experiments

For real-world experiments, data-efficiency is crucial. We perform all experiments in this section relying on a single robot (a single actor) – demonstrating the benefits of RHPO in the low data regime. The performed task is the real world version of the Pile1 task described in Section 4.1. The main task objective is to stack one cube onto a second one and move the gripper away from it. We introduce an additional third cube which serves purely as a distractor.

The setup for the experiments consists of a Sawyer robot arm mounted on a table, equipped with a Robotiq 2F-85 parallel gripper. A basket of size 20cm^2 in front of the robot contains the three cubes. Three cameras on the basket track the cubes using fiducials (augmented reality tags). As in simulation, the agent is provided with proprioception information (joint positions, velocities and torques), a wrist sensor's force and torque readings, as well as the cubes' poses – estimated via the fiducials. The agent action is five dimensional and consists of the three Cartesian translational velocities, the angular velocity of the wrist around the vertical axis and the speed of the gripper's fingers.

Figure 3 plots the learning progress on the real robot for two (out of 7) of the tasks, the simple reach tasks and the stack task – which is the main task of interest. Plots for the learning progress of all tasks are given in the appendix A.6. As can be observed, all methods manage to learn the reach task quickly (within about a few thousand episodes) but only RHPO with a hierarchical policy is able to learn the stacking task (taking about 15 thousand episodes to obtain good stacking success), which takes about 8 days of training on the real robot. With respect to the current state of the baselines and previous experiments it can be estimated that the baselines will take considerably longer time (and we will continue running the 'Independent' baseline experiment after submission until all reach the same training duration).



Reach Stack

Figure 3: Robot Experiments. While simpler tasks such as reaching are learned with comparable efficiency, the later, more complex tasks are acquired significantly faster with a hierarchical policy.

4.2.1 Additional Ablations

In this section we build on the earlier introduced **Pile1** domain to perform a set of ablations, providing additional insights into benefits of RHPO and important factors for robust training.

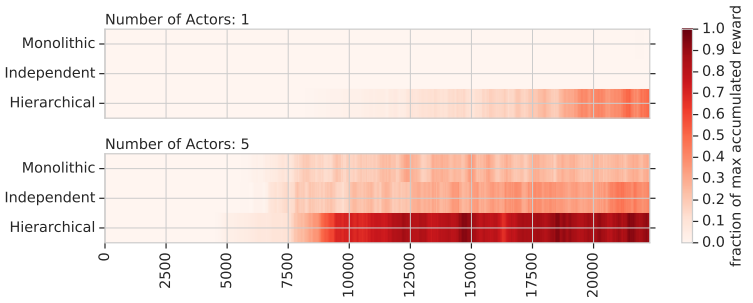


Figure 4: Learning convergence on Pile1 in relation to the number of actors. RHPO results in significantly quicker learning in particular in the low data regime (1-5 actors) where both baselines fail to converge in the given time.

Impact of Data Rate Evaluating in a distributed off-policy setting enables us to investigate the effect of different rates for data generation by controlling the number of actors. Figure 4 demonstrates how the different agents converge slower lower data rates (changing from 5 to 1 actor). These experiments are highly relevant for the application domain as the number of available physical robots for real-world experiments is typically highly limited. To limit computational cost, we focus on the simplest domain from Section 4.1, Pile1, in this comparison. We display the learning curve of the most complicated task within the 7 tasks (the final stack). The results for all tasks as a function of the number of actor processes generating data is available in the Appendix A.8.3.

Importance of Regularization Coordinating convergence progress in hierarchical models can be challenging but can be effectively moderated by the KL constraint. To investigate its importance, we perform an ablation study varying the strength of KL constraints on the high-level controller between prior and the current policy during training – demonstrating a range of possible degenerate behaviors.

As depicted in Figure 5, with a weak KL constraint, the high-level controller can converge too quickly leading to only a single sub-policy getting a gradient signal per step. In addition, the categorical distribution tends to change at a high rate, preventing successful convergence for the low-level policies. On the other hand, the low-level policies are missing task information to encourage decomposition as described in Section 3.2. This fact, in combination with strong KL constraints, can prevent specialization of the low-level policies as the categorical remains near static, finally leading to no or very slow convergence. As long as a reasonable constraint is picked, convergence is fast and the final policies obtain high quality for all tasks. We note that no tuning of the constraints is required across domains and the range of admissible constraints is quite broad.

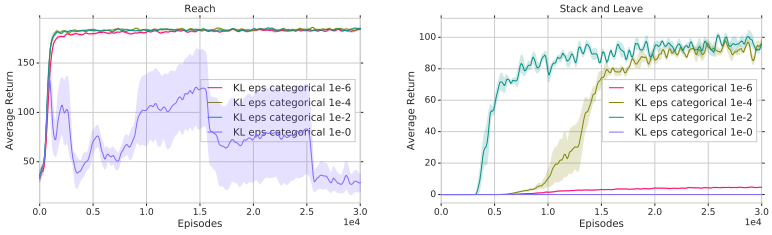


Figure 5: Importance of Categorical KL Constraint. A weak constraint causes the categorical to adapt too fast for the low-level policies a too strong constraint prevents convergence of the categorical.

5 Related Work

Transfer learning, in particular in the multitask context, has long been part of machine learning (ML) for data-limited domains [Caruana, 1997, Torrey and Shavlik, 2010, Pan and Yang, 2010]. Commonly, it is not straightforward to train a single model jointly across different tasks as the solutions to tasks might not only interfere positively but also negatively [Wang et al., 2018].

Preventing this type of forgetting or negative transfer presents a challenge for biological [Singley and Anderson, 1989] as well as artificial systems [French, 1999]. In the context of ML, a common scheme is the reduction of representational overlap [French, 1999, Rusu et al., 2016, Wang et al., 2018]. Bishop [1994] utilize neural networks to parametrize mixture models for representing multi-modal distributions thus mitigating shortcomings of non-hierarchical approaches. Rosenstein et al. [2005] demonstrate that hierarchical classification models can be applied to limit the impact of negative transfer. Our constrained optimization scheme for mixture policies builds on [Abdolmaleki et al., 2018b]. Some aspects of it are similar to Daniel et al. [2016] but we phrase the learning problem in a fashion suitable for large-scale off-policy learning with deep neural networks.

Hierarchical approaches have a long history in the reinforcement learning literature [e.g. Sutton et al., 1999, Dayan and Hinton, 1993]. Similar to our work, the well known options framework [Sutton et al., 1999, Precup, 2000] supports a two level behavior hierarchy, where the higher level chooses from a discrete set of sub-policies or "options" which commonly are run until a termination criterion is satisfied. The framework focuses on the notion of temporal abstraction. A number of works have proposed practical and scalable algorithms for learning option policies with reinforcement learning [e.g. Bacon et al., 2017, Zhang and Whiteson, 2019, Smith et al., 2018] or criteria for option induction [e.g. Harb et al., 2018, Harutyunyan et al., 2019]. Rather than temporal abstraction, RHPO emphasizes the sharing of sub-policies across tasks to provide structure for efficient multitask transfer.

Conceptually different from the options framework and from our work are approaches such as [Vezhnevets et al., 2017, Nachum et al., 2018a,b, Xie et al., 2018] which employ different rewards for different levels of the hierarchy rather than optimizing a single objective for the entire model as we do. Other works have shown the additional benefits for the stability of training and data-efficiency when sequences of high-level actions are given as guidance during optimization in a hierarchical setting [Shiarlis et al., 2018, Andreas et al., 2017, Tirumala et al., 2019].

Probabilistic trajectory models have been used for the discovery of behavior abstractions as part of an end-to-end reinforcement learning paradigm [e.g. Teh et al., 2017, Igl et al., 2019, Tirumala et al., 2019, Galashov et al., 2018] where the models act as learned inductive biases that induce the sharing of behavior across tasks. Several works have previously emphasized the importance of information asymmetry or information hiding [Galashov et al., 2018, Heess et al., 2016]. In a vein similar to the present work [e.g Heess et al., 2016, Tirumala et al., 2019] share a low-level controller across tasks but modulate the low-level behavior via a continuous embedding rather than picking from a small number of mixture components. In related work Hausman et al. [2018], Haarnoja et al. [2018] learn hierarchical policies with continuous latent variables optimizing the entropy regularized objective.

6 Discussion

We introduce a novel framework, RHPO, to enable robust training of hierarchical policies for complex real-world tasks and provide insights into methods for stabilizing the learning process. In simulation as well as on real robots, the proposed approach outperforms baseline methods which either handle tasks independently or utilize implicit sharing. Especially with increasingly complex tasks or limited data rate, as given in real-world applications, we demonstrate hierarchical inductive biases to provide a compelling foundation for transfer learning, reducing the number of environment interactions significantly and often leading to more robust learning as well as improved final performance. While this approach partially mitigates negative interference between tasks in a parallel multitask learning scenario, addressing catastrophic inference in sequential settings remains challenging and provides a valuable direction.

Since with mixture distributions, we are able to marginalize over components when optimizing the weighted likelihood over action samples in Equation 5, the extension towards multiple levels of hierarchy is trivial but can provide a valuable direction for practical future work.

We believe that especially in domains with consistent agent embodiment and high costs for data generation learning tasks jointly and sharing will be imperative. RHPO combines several ideas that we believe will be important: multitask learning with hierarchical and modular policy representations, robust optimization, and efficient off-policy learning. Although we have found this particular combination of components to be very effective we believe it is just one instance of – and step towards – a spectrum of efficient learning architectures that will unlock further applications of RL both in simulation and, importantly, on real hardware.

Acknowledgments

The authors would like to thank Michael Bloesch, Jonas Degrave, Joseph Modayil and Doina Precup for helpful discussion and relevant feedback for shaping our submission. As robotics (and AI research) is a team sport we'd additionally like to acknowledge the support of Francesco Romano, Murilo Martins, Stefano Salicetti, Tom Rothörl and Francesco Nori on the hardware and infrastructure side as well as many others of the DeepMind team.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL `http://arxiv.org/abs/1808.00177`.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

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.

Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *CoRR*, abs/1707.04175, 2017. URL `http://arxiv.org/abs/1707.04175`.

Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in kl-regularized rl. 2018.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. Mutual alignment transfer learning. *arXiv preprint arXiv:1707.07907*, 2017.

Mark K Singley and John Robert Anderson. *The transfer of cognitive skill*. Number 9. Harvard University Press, 1989.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, pages 1–4, 2005.

Christopher M Bishop. Mixture density networks. Technical report, Citeseer, 1994.

Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, Nicolas Heess, and Martin Riedmiller. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018a.

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.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller. Maximum a posteriori policy optimisation. *CoRR*, abs/1806.06920, 2018b.

Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, Razvan Pascanu, Yee Whye Teh, and Nicolas Heess. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv preprint arXiv:1903.07438*, 2019.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1889–1897. JMLR. org, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1406–1415, 2018.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Peter Buchlovsky, David Budden, Dominik Grewe, Chris Jones, John Aslanides, Frederic Besse, Andy Brock, Aidan Clark, Sergio Gómez Colmenarejo, Aedan Pope, Fabio Viola, and Dan Belov. Tf-replicator: Distributed machine learning for researchers. *arXiv preprint arXiv:1902.00465*, 2019.

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*, 2015.

Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. *arXiv preprint arXiv:1811.09751*, 2018.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3 (4):128–135, 1999.

Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *The Journal of Machine Learning Research*, 17(1):3190–3239, 2016.

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

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Shangtong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. *arXiv preprint arXiv:1904.12691*, 2019.

Matthew Smith, Herke Hoof, and Joelle Pineau. An inference-based policy gradient method for learning options. In *International Conference on Machine Learning*, pages 4710–4719, 2018.

Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Rémi Munos, and Doina Precup. The termination critic. *CoRR*, abs/1902.09996, 2019. URL `http://arxiv.org/abs/1902.09996`.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018a.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018b.

Saining Xie, Alexandre Galashov, Siqi Liu, Shaobo Hou, Razvan Pascanu, Nicolas Heess, and Yee Whye Teh. Transferring task goals via hierarchical reinforcement learning, 2018. URL `https://openreview.net/forum?id=S1Y6TtJvG`.

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. Taco: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning*, pages 4661–4670, 2018.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 166–175. JMLR. org, 2017.

Maximilian Igl, Andrew Gambardella, Nantas Nardelli, N Siddharth, Wendelin Böhmer, and Shimon Whiteson. Multitask soft option learning. *arXiv preprint arXiv:1904.01033*, 2019.

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

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 1846–1855, 2018.

Malcolm Reynolds, Gabriel Barth-Maron, Frederic Besse, Diego de Las Casas, Andreas Fidjeland, Tim Green, Adrià Puigdomènech, Sébastien Racanière, Jack Rae, and Fabio Viola. Open sourcing Sonnet - a new library for constructing neural networks. `https://deepmind.com/blog/open-sourcing-sonnet/`, 2017.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

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, 2015.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

A Appendix

A.1 Additional Derivations

In this section we explain the detailed derivations for training hierarchical policies parameterized as a mixture of Gaussians.

A.1.1 Obtaining Non-parametric Policies

In each policy improvement step, to obtain non-parametric policies for a given state and task distribution, we solve the following program:

$$\begin{aligned}
&\max_q \mathbb{E}_{\mu(s),i\sim I}\left[\mathbb{E}_{q(a|s,i)}\left[\hat{Q}(s,a,i)\right]\right] \\
&s.t. \mathbb{E}_{\mu(s),i\sim I}\left[\mathrm{KL}(q(\cdot|s,i),\pi(\cdot|s,i,\theta_t))\right] < \epsilon \\
&s.t. \mathbb{E}_{\mu(s),i\sim I}\left[\mathbb{E}_{q(a|s)}\left[1\right]\right] = 1.
\end{aligned}$$

To make the following derivations easier to follow we open up the expectations, writing them as integrals explicitly. For this purpose let us define the joint distribution over states $s \sim \mu(s)$ together with randomly sampled tasks $i \sim I$ as $\mu(s,i) = p(s|\mathcal{D})\mathcal{U}(i \in I)$, where $\mathcal{U}$ denotes the uniform distribution over possible tasks. This allows us to re-write the expectations that include the corresponding distributions, i.e. $\mathbb{E}_{\mu(s),i\sim I}[1] = \mathbb{E}_{\mu(s,i)}[1]$, but again, note that i here is not necessarily the task under which s was observed. We can then write the Lagrangian equation corresponding to the above described program as

$$
\begin{aligned}
L(q,\eta,\gamma) = &\int \mu(s,i) \int q(a|s,i)\hat{Q}(s,a,i)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i \\
&+ \eta\left(\epsilon - \int \mu(s,i) \int q(a|s,i) \log \frac{q(a|s,i)}{\pi(a|s,i,\theta_t)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i}\right) \\
&+ \gamma\left(1 - \iint \mu(s,i) q(a|s,i)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i\right).
\end{aligned}
$$

Next we maximize the Lagrangian L w.r.t the primal variable q. The derivative w.r.t q reads,

$$
\begin{aligned}
\partial q L(q,\eta,\gamma) = &\hat{Q}(a,s,i) - \eta \log q(a|s,i) \\
&+ \eta \log \pi(a|s,i,\theta_t) - \eta - \gamma.
\end{aligned}
$$

Setting it to zero and rearranging terms we obtain

$$
q(a|s,i) = \pi(a|s,i,\theta_t) \exp\left(\frac{\hat{Q}(s,a,i)}{\eta}\right) \exp\left(-\frac{\eta+\gamma}{\eta}\right).
$$

However, the last exponential term is a normalization constant for q. Therefore we can write,

$$
\begin{aligned}
\exp\left(\frac{\eta+\gamma}{\eta}\right) &= \int \pi(a|s,i,\theta_t) \exp\left(\frac{Q(s,a,i)}{\eta}\right) \mathrm{d}a \\
\frac{\eta+\gamma}{\eta} &= \log\left(\int \pi(a|s,i,\theta_t) \exp\left(\frac{Q(s,a,i)}{\eta}\right) \mathrm{d}a\right). \qquad (7)
\end{aligned}
$$

Now, to obtain the dual function $g(\eta)$, we plug in the solution to the KL constraint term (second term) of the Lagrangian which yields

$$
\begin{aligned}
L(q,\eta,\gamma) = &\int \mu(s,i) \int q(a|s,i) Q(s,a,i)\,\mathrm{d}a\,\mathrm{d}s \\
&+ \eta\left(\epsilon - \int \mu(s,i) \int q(a|s,i) \log \frac{\pi(a|s,i,\theta_t) \exp\left(\frac{Q(s,a,i)}{\eta}\right) \exp\left(-\frac{\eta+\gamma}{\eta}\right)}{\pi(a|s,i,\theta_t)}\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i\right) \\
&+ \gamma\left(1 - \iint \mu(s,i) q(a|s,i)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i\right).
\end{aligned}
$$

After expanding and rearranging terms we get

$$
\begin{aligned}
L(q,\eta,\gamma) = &\int \mu(s,i) \int q(a|s,i) Q(s,a,i)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i \\
&- \eta \int \mu(s,i) \int q(a|s,i)\Big[\frac{Q(s,a,i)}{\eta} + \log \pi(a|s,i;\theta_t) - \frac{\eta+\gamma}{\eta}\Big]\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i \\
&+ \eta\epsilon + \eta \int \mu(s,i) \int q(a|s,i) \log \pi(a|s,i;\theta_t)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i \\
&+ \gamma\left(1 - \iint \mu(s,i) q(a|s,i)\,\mathrm{d}a\,\mathrm{d}s\,\mathrm{d}i\right).
\end{aligned}
$$

Most of the terms cancel out and after rearranging the terms we obtain

$$
L(q,\eta,\gamma) = \eta\epsilon + \eta \int \mu(s,i) \frac{\eta+\gamma}{\eta}\,\mathrm{d}s\,\mathrm{d}i.
$$

Note that we have already calculated the term inside the integral in Equation 7. By plugging in equation 7 we obtain the dual function

$$
g(\eta) = \eta\epsilon + \eta \int \mu(s,i) \log\left(\int \pi(a|s,i,\theta_t) \exp\left(\frac{Q(s,a,i)}{\eta}\right) \mathrm{d}a\right) \mathrm{d}s\,\mathrm{d}i, \qquad (8)
$$

which we can minimize with respect to η based on samples from the replay buffer.

A.1.2 Extended Update Rules For Fitting a Mixture of Gaussians

After obtaining the non parametric policies, we fit a parametric policy to samples from said non-parametric policies – effectively employing using maximum likelihood estimation with additional regularization based on a distance function $\mathcal{T}$, i.e,

$$\begin{aligned}\theta^{(k+1)} &= \arg\min_{\theta} \mathbb{E}_{s\sim\mathcal{D},i\sim I}\Big[\mathrm{KL}\big(q_k(\cdot|s,i)\|\pi_\theta(\cdot|s,i)\big)\Big] \\ &= \arg\min_{\theta} \mathbb{E}_{s\sim\mathcal{D},i\sim I,a\sim q(\cdot|s,i)}\Big[-\log\pi_\theta(a|s,i)\Big], \\ &= \arg\max_{\theta} \mathbb{E}_{s\sim\mathcal{D},i\sim I,a\sim q(\cdot|s,i)}\Big[\log\pi_\theta(a|s,i)\Big], \\ &\text{s.t. } \mathbb{E}_{s\sim\mathcal{D},i\sim I}\Big[\mathcal{T}(\pi_{\theta_k}(\cdot|s,i)|\pi_\theta(\cdot|s,i))\Big] < \epsilon,\end{aligned} \tag{9}$$

where $\mathcal{D}$ is an arbitrary distance function to evaluate the change of the new policy with respect to a reference/old policy, and ϵ denotes the allowed change for the policy. To make the above objective amenable to gradient based optimization we employ Lagrangian Relaxation, yielding the following primal:

$$\max_{\theta}\min_{\alpha>0} L(\theta,\alpha) = \mathbb{E}_{s\sim\mathcal{D},i\sim I,a\sim q(\cdot|s,i)}\Big[\log\pi_\theta(a|s,i)\Big] + \alpha\Big(\epsilon - \mathbb{E}_{s\sim\mathcal{D},,i\sim I}\big[\mathcal{T}(\pi_{\theta_k}(\cdot|s,i),\pi_\theta(\cdot|s,i))\big]\Big). \tag{10}$$

We solve for θ by iterating the inner and outer optimization programs independently: We fix the parameters θ to their current value and optimize for the Lagrangian multipliers (inner minimization) and then we fix the Lagrangian multipliers to their current value and optimize for θ (outer maximization). In practice we found that it is effective to simply perform one gradient step each in inner and outer optimization for each sampled batch of data.

The optimization given above is general, i.e. it works for any general type of policy. As described in the main paper, we consider hierarchical policies of the form

$$\pi_\theta(a|s,i) = \sum_{o=1}^{M} \pi_\theta^L(a|s,o)\,\pi_\theta^H(o|s,i)\,. \tag{11}$$

In particular, in all experiments we made use of a mixture of Gaussians parametrization, where the high level policy π_θ^H is a categorical distribution over low level π_θ^L Gaussian policies, i.e,

$$\pi_\theta(a|s,i) = \pi(a|\{\alpha_\theta^j,\mu_\theta^j,\Sigma_\theta^j\}(s)_{j=1\dots M}) = \sum_{j=1}^{M}\alpha_\theta^j(s,i)\mathcal{N}(\mu_\theta^j(s),\Sigma_\theta^j(s))$$
$$\forall s \sum_{j=1}^{M}\alpha^j(s,i) = 1, \text{ and, } \alpha^j(s,i) > 0$$

where j denote the index of components and α is the high level policy π^H assigning probabilities to each mixture component for a state s given the task and the low level policies are all Gaussian. Here α^js are the probabilities for a categorical distribution over the components.

We also define the following distance function between old and new mixture of Gaussian policies

$$\mathcal{T}(\pi_{\theta_k}(\cdot|s,i),\pi_\theta(\cdot|s,i)) = \mathcal{T}_H(s,i) + \mathcal{T}_L(s)$$

$$\mathcal{T}_H(s,i) = \mathrm{KL}(\mathrm{Cat}(\{\alpha_{\theta_k}^j(s,i)\}_{j=1\dots M})\|\mathrm{Cat}(\{\alpha_\theta^j(s,i)\}_{j=1\dots M}))$$

$$\mathcal{T}_L(s) = \frac{1}{M}\sum_{j=1}^{M}\mathrm{KL}(\mathcal{N}(\mu_{\theta_k}^j(s),\Sigma_{\theta_k}^j(s))\|\mathcal{N}(\mu_\theta^j(s),\Sigma_\theta^j(s)))$$

where $\mathcal{D}_H$ evaluates the KL between categorical distributions and $\mathcal{D}_L$ corresponds to the average KL across Gaussian components, as also described in the main paper (c.f. Equation 5 in the main paper).

In order to bound the change of categorical distributions, means and covariances of the components independently – which makes it easy to control the convergence of the policy and which can prevent premature convergence as argued in Abdolmaleki et al. [2018a] – we separate out the following three intermediate policies

$$\begin{aligned}
\pi_\theta^\Sigma(a|s,i) &= \pi(a|\{\alpha_{\theta_k}^j, \mu_{\theta_k}^j, \Sigma_\theta^j\}(s,i)_{j=1\dots M}) \\
\pi_\theta^\mu(a|s,i) &= \pi(a|\{\alpha_{\theta_k}^j, \mu_\theta^j, \Sigma_{\theta_k}^j\}(s,i)_{j=1\dots M}) \\
\pi_\theta^\alpha(a|s,i) &= \pi(a|\{\alpha_\theta^j, \mu_{\theta_k}^j, \Sigma_{\theta_k}^j\}(s,i)_{j=1\dots M})
\end{aligned}$$

Which yields the following final optimization program

$$\begin{aligned}
\theta^{(k+1)} = \arg\max_\theta & \mathbb{E}_{s\sim\mathcal{D}, i\sim I, a\sim q(\cdot|s,i)}\Big[\log \pi_\theta^\mu(a|s,i) + \log \pi_\theta^\Sigma(a|s,i) + \log \pi_\theta^\alpha(a|s,i)\Big], \\
& \text{s.t. } \mathbb{E}_{s\sim\mathcal{D}, i\sim I}\Big[\mathcal{T}(\pi_{\theta_k}(a|s,i)|\pi_\theta^\mu(a|s,i))\Big] < \epsilon_\mu, \\
& \text{s.t. } \mathbb{E}_{s\sim\mathcal{D}, i\sim I}\Big[\mathcal{T}(\pi_{\theta_k}(a|s,i)|\pi_\theta^\Sigma(a|s,i))\Big] < \epsilon_\Sigma, \\
& \text{s.t. } \mathbb{E}_{s\sim\mathcal{D}, i\sim I}\Big[\mathcal{T}(\pi_{\theta_k}(a|s,i)|\pi_\theta^\alpha(a|s,i))\Big] < \epsilon_\alpha,
\end{aligned} \tag{12}$$

This decoupling allows us to set different ϵ values for the change of means, covariances and categorical probabilities, i.e., $\epsilon_\mu, \epsilon_\Sigma, \epsilon_\alpha$. Different ϵ lead to different learning rates. We always set a much smaller epsilon for the covariance and categorical than for the mean. The intuition is that while we would like the distribution to converge quickly in the action space, we also want to keep the exploration both locally (via the covariance matrix) and globally (via the high level categorical distribution) to avoid premature convergence.

A.2 Algorithmic Details

A.2.1 Pseudocode for the full procedure

We provide a pseudo-code listing of the full optimization procedure – and the asynchronous data gathering – performed by RHPO in Algorithm 1 and 2. The implementation relies on Sonnet [Reynolds et al., 2017] and TensorFlow [Abadi et al., 2015].

A.2.2 Details on the policy improvement step

As described in the main paper we consider the same setting as scheduled auxiliary control setting (SAC-X) [Riedmiller et al., 2018] to perform policy improvement (with uniform random switches between tasks every N steps within an episode, the SAC-U setting).

Given a replay buffer containing data gathered from all tasks, where for each trajectory snippet $\tau = \{(s_0, a_0, R_0), \dots, (s_L, a_L, R_L)\}$ we record the rewards for all tasks $R_t = [r_{i_1}(s_t, a_t), \dots, r_{i_{|I|}}(s_t, a_t)]$ as a vector in the buffer, we define the retrace objective for learning $\hat{Q}$, parameterized via ϕ, following Riedmiller et al. [2018] as

$$\begin{aligned}
&\min_\phi L(\phi) = \sum_{i\sim I} \mathbb{E}_{\tau\sim\mathcal{D}}\Big[\big(r_i(s_t,a_t) + \gamma Q^{ret}(s_{t+1}, a_{t+1}, i) - \hat{Q}_\phi(s_t, a_t, i)\big)^2\Big], \text{with} \\
&Q^{\text{ret}}(s_t, a_t, i) = \sum_{j=t}^{\infty}\Big(\gamma^{j-t}\prod_{k=t}^{j} c_k\Big)\Big[\delta_Q(s_j, s_{j+1})\Big], \\
&\delta_Q(s_j, s_{j+1}) = r_i(s_j, a_j) + \gamma \mathbb{E}_{\pi_{\theta_k}(a|s_{j+1},i)}[\hat{Q}_{\phi'}(s_{j+1}, \cdot, i; \phi')] - \hat{Q}_{\phi'}(s_j, a_j, i),
\end{aligned} \tag{13}$$

where the importance weights are defined as $c_k = \min\left(1, {}^{\pi_{\theta_k}(a_k|s_k,i)}/_{b(a_k|s_k)}\right)$, with $b(a_k|s_k)$ denoting an arbitrary behavior policy; in particular this will be the policy for the executed tasks during an

Algorithm 1 Asynchronous Learner

Input: N_{steps} number of update steps, $N_{\text{targetUpdate}}$ update steps between target update, N_s number of action samples per state, KL regularization parameters ϵ, initial parameters for π, η and ϕ
initialize N = 0
while $k \leq N_{\text{steps}}$ **do**
  for k in $[0...N_{\text{targetUpdate}}]$ **do**
    sample a batch of trajectories τ from replay buffer B
    sample N_s actions from π_{θ_k} to estimate expectations below
    // compute mean gradients over batch for policy, Lagrangian multipliers and Q-function
    $\delta_\pi \leftarrow -\nabla_\theta \sum_{s_t \in \tau} \sum_{j=1}^{N_s} \exp\left(\frac{Q(s_t, a_j, i)}{\eta}\right) \log \pi_\theta(a_j | s_t, i)$ following Eq. 5
    $\delta_\eta \leftarrow \nabla_\eta g(\eta) = \nabla_\eta \eta\epsilon + \eta \sum_{s_t \in \tau} \log \frac{1}{N_s} \sum_{j=1}^{N_s} \exp\left(\frac{Q(s_t, a_j, i)}{\eta}\right)$ following Eq. 8
    $\delta_Q \leftarrow \nabla_\phi \sum_{i \sim I} \sum_{(s_t, a_t) \in \tau} \left(\hat{Q}_\phi(s_t, a_t, i) - Q^{\text{ret}}\right)^2$ with Q^{ret} following Eq. 6
    // apply gradient updates
    $\pi_{\theta_{k+1}} =$ optimizer_update(π, δ_π),
    $\eta =$ optimizer_update(η, δ_η)
    $\hat{Q}_\phi =$ optimizer_update($\hat{Q}_\phi$, δ_Q)
    $k = k + 1$
  end for
  // update target networks
  $\pi' = \pi$, $Q' = Q$
end while

Algorithm 2 Asynchronous Actor

Input: $N_{\text{trajectories}}$ number of total trajectories requested, T steps per episode, ξ scheduling period
initialize N = 0
while $N < N_{\text{trajectories}}$ **do**
  fetch parameters θ
  // collect new trajectory from environment
  $\tau = \{\}$
  for t in $[0...T]$ **do**
    if $t \pmod{\xi} \equiv 0$ **then**
      // sample active task from uniform distribution
      $i_{act} \sim I$
    end if
    $a_t \sim \pi_\theta(\cdot | s_t, i_{act})$
    // execute action and determine rewards for all tasks
    $\bar{r} = [r_{i_1}(s_t, a_t), \ldots, r_{i_{|I|}}(s_t, a_t)]$
    $\tau \leftarrow \tau \cup \{(s_t, a_t, \bar{r}, \pi_\theta(a_0 | s_t, i_{act}))\}$
  end for
  send batch trajectories τ to replay buffer
  $N = N + 1$
end while

episode as in [Riedmiller et al., 2018]. Note that, in practice, we truncate the infinite sum after L steps, bootstrapping with $\hat{Q}$. We further perform optimization of Equation (6) via gradient descent and make use of a target network [Mnih et al., 2015], denoted with parameters ϕ', which we copy from ϕ after a couple of gradient steps. We reiterate that, as the state-action value function $\hat{Q}$ remains independent of the policy's structure, we are able to utilize any other off-the-shelf Q-learning algorithm such as TD(0) [Sutton, 1988].

Given that we utilize the same policy evaluation mechanism as SAC-U it is worth pausing here to identify the differences between SAC-U and our approach. The main difference is in the policy parameterization: SAC-U used a monolithic policy for each task $\pi(a|s, i)$ (although a neural network with shared components, potentially leading to some implicit task transfer, was used). Furthermore, we perform policy optimization based on MPO instead of using stochastic value gradients (SVG [Heess et al., 2016]). We can thus recover a variant of plain SAC-U using MPO if we drop the hierarchical policy parameterization, which we employ in the single task experiments in the main paper.

A.2.3 Network Architectures

To represent the Q-function in the multitask case we use the network architecture from SAC-X (see right sub-figure in Figure 6). The proprioception of the robot, the features of the objects and the actions are fed together in a torso network. At the input we use a fully connected first layer of 200 units, followed by a layer normalization operator, an optional tanh activation and another fully connected layer of 200 units with an ELU activation function. The output of this torso network is shared by independent head networks for each of the tasks (or intentions, as they are called in the SAC-X paper). Each head has two fully connected layers and outputs a Q-value for this task, given the input of the network. Using the task identifier we then can compute the Q value for a given sample by discrete selection of the according head output.

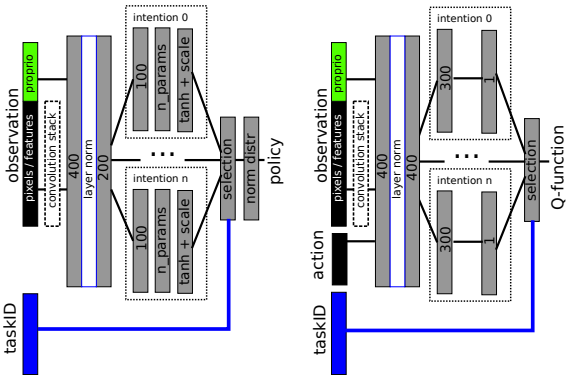


Figure 6: Schematics of the fully connected networks in SAC-X. While we use the Q-function (right sub-figure) architecture in all multitask experiments, we investigate variations of the policy architecture (left sub-figure) in this paper (see Figure 7).

While we use the network architecture for the Q function for all multitask experiments, we investigate different architectures for the policy in this paper. The original SAC-X policy architecture is shown in Figure 6 (left sub-figure). The main structure follows the same basic principle that we use in the Q function architecture. The only difference is that the heads compute the required parameters for the policy distribution we want to use (see subsection A.2.4). This architecture is referenced as the independent heads (or task dependent heads).

The alternatives we investigate in this paper are the monolithic policy architecture (see Figure 7, left sub-figure) and the hierarchical policy architecture (see Figure 7, right sub-figure). For the monolithic policy architecture we reduce the original policy architecture basically to one head and append the task-id as a one-hot encoded vector to the input. For the hierarchical architecture, we build on the same torso and create a set of networks parameterizing the Gaussians which are shared across tasks and a task-specific network to parameterize the categorical distribution for each task.

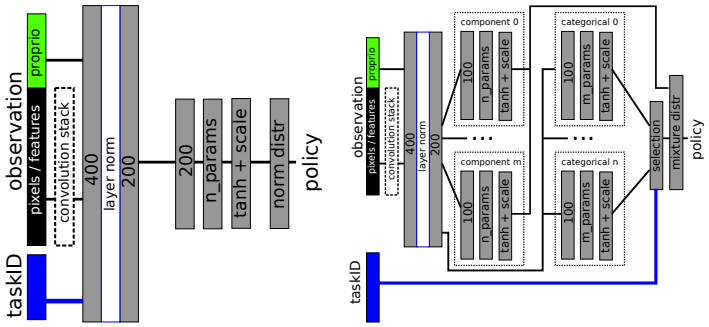


Figure 7: Schematics of the alternative multitask policy architectures used in this paper. Left sub-figure: the monolithic architecture; Right sub-figure: the hierarchical architecture.

Hyperparameters	Hierarchical	Single Gaussian
Policy net	200-200-200	200-200-200
Number of actions sampled per state	10	10
Q function net	500-500-500	500-500-500
Number of components	3	NA
ϵ	0.1	0.1
ϵ_μ	0.0005	0.0005
ϵ_Σ	0.00001	0.00001
ϵ_α	0.0001	NA
Discount factor (γ)	0.99	0.99
Adam learning rate	0.0002	0.0002
Replay buffer size	2000000	2000000
Target network update period	250	250
Batch size	256	256
Activation function	elu	elu
Layer norm on first layer	Yes	Yes
Tanh on output of layer norm	Yes	Yes
Tanh on input actions to Q-function	Yes	Yes
Retrace sequence length	10	10

Table 1: Hyperparameters - Single Task

The final mixture distribution is task-dependent for the high-level controller but task-independent for the low-level policies.

A.2.4 Algorithm Hyperparameters

In this section we outline the details on the hyperparameters used for RHPO and baselines in both single task and multitask experiments. All experiments use feed-forward neural networks. We consider a flat policy represented by a Gaussian distribution and a hierarchical policy represented by a mixture of Gaussians distribution. The flat policy is given by a Gaussian distribution with a diagonal covariance matrix, i.e, $\pi(a|s,\theta) = \mathcal{N}(\mu, \Sigma)$. The neural network outputs the mean $\mu = \mu(s)$ and diagonal Cholesky factors $A = A(s)$, such that $\Sigma = AA^T$. The diagonal factor A has positive diagonal elements enforced by the softplus transform $A_{ii} \leftarrow \log(1 + \exp(A_{ii}))$ to ensure positive definiteness of the diagonal covariance matrix. Mixture of Gaussian policy has a number of Gaussian components as well as a categorical distribution for selecting the components. The neural network outputs the Gaussian components based on the same setup described above for a single Gaussian and outputs the logits for representing the categorical distribution. Tables 1 show the hyperparameters we used for the single tasks experiments. We found layer normalization and a hyperbolic tangent ($tanh$) on the layer following the layer normalization are important for stability of the algorithms. For RHPO the most important hyperparameters are the constraints in Step 1 and Step 2 of the algorithm.

Hyperparameters	Hierarchical	Independent	Monolith
Policy torso (shared across tasks)	400-200		
Policy task-dependent heads	100 (high-level controller)	100	NA
Policy shared heads	100 (per mixture component)	NA	200
Number of action samples	20		
Q function torso (shared across tasks)	400-400		
Q function head (per task)	300		
Number of components	number of tasks	NA	NA
Discount factor (γ)	0.99		
Replay buffer size	1e6 * number of tasks		
Target network update period	500		
Batch size	256 (512 for Pile1)		

Table 2: Hyperparameters - Multitask. Values are taken from the single task experiments with the above mentioned changes.

A.3 Additional Details on the SAC-U with SVG baseline

For the SAC-U baseline we used a re-implementation of the method from [Riedmiller et al., 2018] using SVG [Heess et al., 2015] for optimizing the policy. Concretely we use the same basic network structure as for the "Monolithic" baseline with MPO and parameterize the policy as

$$\pi_\theta = \mathcal{N}(\mu_\theta(s,i), \sigma_\theta^2(s,i)I),$$

where I denotes the identity matrix and $\sigma_\theta(s,i)$ is computed from the network output via a softplus activation function.

Together with entropy regularization, as described in [Riedmiller et al., 2018] the policy can be optimized via gradient ascent, following the reparameterized gradient for a given states sampled from the replay:

$$\nabla_\theta \mathbb{E}_{\pi_\theta(a|s,i)}[\hat{Q}(a,s,i)] + \alpha \mathrm{H}\Big(\pi_\theta(a|s,i)\Big), \tag{14}$$

which can be computed, using the reparameterization trick, as

$$\mathbb{E}_{\zeta\sim\mathcal{N}(0,I)}[\nabla_\theta g_\theta(s,i,\zeta)\nabla_g Q(g_\theta(s,i,\zeta),s,i)] + \alpha\nabla_\theta \mathrm{H}\Big(\pi_\theta(a|s,i)\Big), \tag{15}$$

where $g_\theta(s,i,\zeta) = \mu_\theta(s,i) + \sigma_\theta(\mathbf{s}) * \zeta$ is now a deterministic function of a sample from the standard multivariate normal distribution. See e.g. Heess et al. [2015] (for SVG) as well as Kingma and Welling [2014] (for the reparameterization trick) for a detailed explanation.

A.4 Details on the Experimental Setup

A.4.1 Simulation (Single- and Multitask)

For the simulation of the robot arm experiments the numerical simulator MuJoCo [3] was used – using a model we identified from the real robot setup.

We run experiments of length 2 - 7 days for the simulation experiments (depending on the task) with access to 2-5 recent CPUs with 32 cores each (depending on the number of actors) and 2 recent NVIDIA GPUs for the learner. Computation for data buffering is negligible.

A.4.2 Real Robot Multitask

Compared to simulation where the ground truth position of all objects is known, in the real robot setting, three cameras on the basket track the cube using fiducials (augmented reality tags).

[3] MuJoCo: see www.mujoco.org

Table 3: Action space for the Sawyer experiments.

Entry	Dimensions	Unit	Range
Translational Velocity in x, y, z	3	m/s	[-0.07, 0.07]
Wrist Rotation Velocity	1	rad/s	[-1, 1]
Finger speed	1	tics/s	[-255, 255]

For safety reasons, external forces are measured at the wrist and the episode is terminated if a threshold of 20N on any of the three principle axes is exceeded (this is handled as a terminal state with reward 0 for the agent), adding further to the difficulty of the task.

The real robot setup differs from the simulation in the reset behaviour between episodes, since objects need to be physically moved around when randomizing, which takes a considerable amount of time. To keep overhead small, object positions are randomized only every 25 episodes, using a hand-coded controller. Objects are also placed back in the basket if they were thrown out during the previous episode. Other than that, objects start in the same place as they were left in the previous episode. The robot's starting pose is randomized each episode, as in simulation.

A.5 Task Descriptions

A.5.1 Pile1

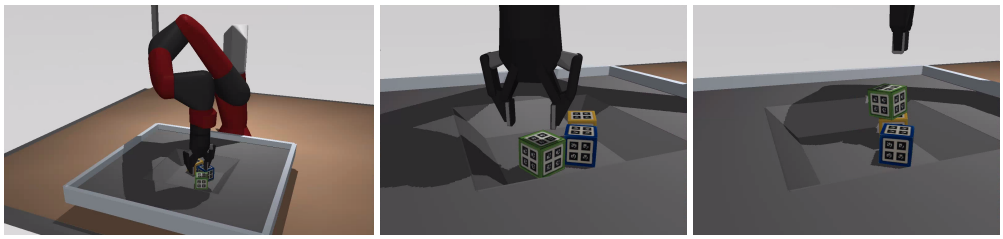


Figure 8: Sawyer Set-Up.

For this task we have a real setup and a MuJoCo simulation that are well aligned. It consists of a Sawyer robot arm mounted on a table and equipped with a Robotiq 2F-85 parallel gripper. In front of the robot there is a basket of size 20x20 cm which contains three cubes with an edge length of 5 cm (see Figure 8).

The agent is provided with proprioception information for the arm (joint positions, velocities and torques), and the tool center point position computed via forward kinematics. For the gripper, it receives the motor position and velocity, as well as a binary grasp flag. It also receives a wrist sensor's force and torque readings. Finally, it is provided with the cubes' poses as estimated via the fiducials, and the relative distances between the arm's tool center point and each object. At each time step, a history of two previous observations is provided to the agent, along with the last two joint control commands, in order to account for potential communication delays on the real robot. The observation space is detailed in Table 4.

The robot arm is controlled in Cartesian mode at 20Hz. The action space for the agent is 5-dimensional, as detailed in Table 3. The gripper movement is also restricted to a cubic volume above the basket using virtual walls.

For the Pile1 experiment we use 7 different task to learn, following the SAC-X principles. The first 6 tasks are seen as auxiliary tasks that help to learn the final task (STACK_AND_LEAVE(G, Y)) of stacking the green cube on top of the yellow cube. Overview of the used tasks:

- *REACH(G)*: $stol(d(TCP, G), 0.02, 0.15)$:
  Minimize the distance of the TCP to the green cube.

Table 4: Observation used in the experiments with the Sawyer arm. An object's pose is represented as its world coordinate position and quaternion. In the table, m denotes meters, rad denotes radians, and q refers to a quaternion in arbitrary units (au).

Entry	Dimensions	Unit
Joint Position (Arm)	7	rad
Joint Velocity (Arm)	7	rad/s
Joint Torque (Arm)	7	Nm
Joint Position (Hand)	1	rad
Joint Velocity (Hand)	1	tics/s
Force-Torque (Wrist)	6	N, Nm
Binary Grasp Sensor	1	au
TCP Pose	7	m, au
Last Control Command (Joint Velocity)	8	rad/s
Green Cube Pose	7	m, au
Green Cube Relative Pose	7	m, au
Yellow Cube Pose	7	m, au
Yellow Cube Relative Pose	7	m, au
Blue Cube Pose	7	m, au
Blue Cube Relative Pose	7	m, au

- *GRASP*:
  Activate grasp sensor of gripper ("inward grasp signal" of Robotiq gripper)
- *LIFT(G)*: $slin(G, 0.03, 0.10)$
  Increase z coordinate of an object more than 3cm relative to the table.
- *PLACE_WIDE(G, Y)*: $stol(d(G, Y + [0, 0, 0.05]), 0.01, 0.20)$
  Bring green cube to a position 5cm above the yellow cube.
- *PLACE_NARROW(G, Y)*: $stol(d(G, Y + [0, 0, 0.05]), 0.00, 0.01)$:
  Like PLACE_WIDE(G, Y) but more precise.
- *STACK(G, Y)*: $btol(d_{xy}(G, Y), 0.03) * btol(d_z(G, Y) + 0.05, 0.01) * (1 - \textit{GRASP})$
  Sparse binary reward for bringing the green cube on top of the yellow one (with 3cm tolerance horizontally and 1cm vertically) and disengaging the grasp sensor.
- *STACK_AND_LEAVE(G, Y)*: $stol(d_z(TCP, G) + 0.10, 0.03, 0.10) * \textit{STACK(G, Y)}$
  Like STACK(G, Y), but needs to move the arm 10cm above the green cube.

Let $d(o_i, o_j)$ be the distance between the reference of two objects (the reference of the cubes are the center of mass, TCP is the reference of the gripper), and let d_A be the distance only in the dimensions denoted by the set of axes A. We can define the reward function details by:

$$stol(v, \epsilon, r) = \begin{cases} 1 & \text{iff } |v| < \epsilon \\ 1 - tanh^2(\frac{atanh(\sqrt{0.95})}{r}|v|) & \text{else} \end{cases} \tag{16}$$

$$slin(v, \epsilon_{min}, \epsilon_{max}) = \begin{cases} 0 & \text{iff } v < \epsilon_{min} \\ 1 & \text{iff } v > \epsilon_{max} \\ \frac{v - \epsilon_{min}}{\epsilon_{max} - \epsilon_{min}} & \text{else} \end{cases} \tag{17}$$

$$btol(v, \epsilon) = \begin{cases} 1 & \text{iff } |v| < \epsilon \\ 0 & \text{else} \end{cases} \tag{18}$$

A.5.2 Pile2

For the *Pile2* task, taken from Riedmiller et al. [2018], we use a different robot arm, control mode and task setup to emphasize that RHPO's improvements are not restricted to cartesian control or a specific robot and that the approach also works for multiple external tasks.

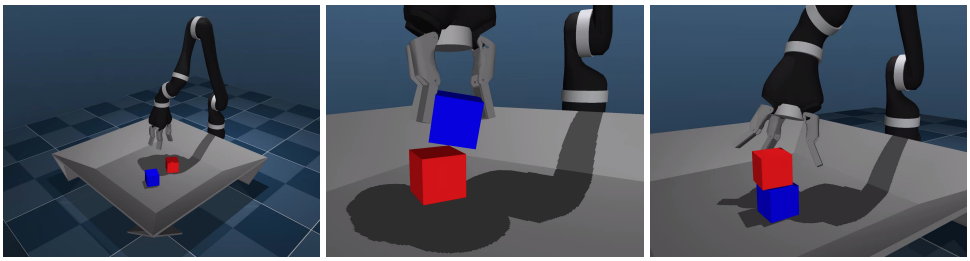


Figure 9: The *Pile2* set-up in simulation with two main tasks: The first is to stack the blue on the red cube, the second is to stack the red on the blue cube.

Table 5: Action space used in the experiments with the Kinova Jaco Arm.

Entry	Dimensions	Unit	Range
Joint Velocity (Arm)	6	rad/sec	[-0.8, 0.8]
Joint Velocity (Hand)	3	rad/sec	[-0.8, 0.8]

Here, the agent controls a simulated Kinova Jaco robot arm, equipped with a Kinova KG-3 gripper. The robot faces a 40 x 40 cm basket that contains a red cube and a blue cube. Both cubes have an edge length of 5 cm (see Figure 9). The agent is provided with proprioceptive information for the arm and the fingers (joint positions and velocities) as well as the tool center point position (TCP) computed via forward kinematics. Further, the simulated gripper is equipped with a touch sensor for each of the three fingers, whose value is provided to the agent as well. Finally, the agent receives the cubes' poses, their translational and rotational velocities and the relative distances between the arm's tool center point and each object. Neither observation nor action history is used in the *Pile2* experiments. The cubes are spawned at random on the table surface and the robot hand is initialized randomly above the table-top with a height offset of up to 20 cm above the table (minimum 10 cm). The observation space is detailed in Table 6.

The robot arm is controlled in raw joint velocity mode at 20 Hz. The action space is 9-dimensional as detailed in Table 5. There are no virtual walls and the robot's movement is solely restricted by the velocity limits and the objects in the scene.

Analogous to *Pile1* and the SAC-X setup, we use 10 different task for *Pile2*. The first 8 tasks are seen as auxiliary tasks, that the agent uses to learn the main *two* tasks *PILE_RED* and *PILE_BLUE*, which represent stacking the red cube on the blue cube and stacking the blue cube on the red cube respectively. The tasks used in the experiment are:

- *REACH(R)* = $stol(d(TCP, R), 0.01, 0.25)$:
  Minimize the distance of the TCP to the red cube.
- *REACH(B)* = $stol(d(TCP, B), 0.01, 0.25)$:
  Minimize the distance of the TCP to the blue cube.
- *MOVE(R)* = $slin(|\, linvel(R)\,|, 0, 1)$:
  Move the red cube.
- *MOVE(B)* = $slin(|\, linvel(B)\,|, 0, 1)$:
  Move the blue cube.
- *LIFT(R)* = $btol(pos_z(R), 0.05)$
  Increase the z-coordinate of the red cube to more than 5cm relative to the table.
- *LIFT(B)* = $btol(pos_z(B), 0.05)$
  Increase the z-coordinate of the blue cube to more than 5cm relative to the table.
- *ABOVE_CLOSE(R, B)* = $above(R, B) * stol(d(R, B), 0.05, 0.2)$
  Bring the red cube to a position above of and close to the blue cube.
- *ABOVE_CLOSE(B, R)* = $above(B, R) * stol(d(R, B), 0.05, 0.2)$
  Bring the blue cube to a position above of and close to the red cube.

Table 6: Observation used in the experiments with the Kinova Jaco Arm. An object's pose is represented as its world coordinate position and quaternion. The lid position and velocity are only used in the *Clean-Up* task. In the table, m denotes meters, rad denotes radians, and q refers to a quaternion in arbitrary units (au).

Entry	Dimensions	Unit
Joint Position (Arm)	6	rad
Joint Velocity (Arm)	6	rad/s
Joint Position (Hand)	3	rad
Joint Velocity (Hand)	3	rad/s
TCP Position	3	m
Touch Force (Fingers)	3	N
Red Cube Pose	7	m, au
Red Cube Velocity	6	m/s, dq/dt
Red Cube Relative Position	3	m
Blue Cube Pose	7	m, au
Blue Cube Velocity	6	m/s, dq/dt
Blue Cube Relative Position	3	m
Lid Position	1	rad
Lid Velocity	1	rad/s

- *PILE(R)*:
  Place the red cube on another object (touches the top). Only given when the cube doesn't touch the robot or the table.
- *PILE(B)*:
  Place the blue cube on another object (touches the top). Only given when the cube doesn't touch the robot or the table.

The sparse reward *above(A, B)* is given by comparing the bounding boxes of the two objects *A* and *B*. If the bounding box of object A is completely above the highest point of object B's bounding box, *above(A, B)* is 1, otherwise *above(A, B)* is 0.

A.5.3 Clean-Up

The *Clean-Up* task is also taken from Riedmiller et al. [2018] and builds on the setup described for the *Pile2* task. Besides the two cubes, the work-space contains an additional box with a moveable lid, that is always closed initially (see Figure 10). The agent's goal is to clean up the scene by placing the cubes inside the box. In addition to the observations used in the *Pile2* task, the agent observes the lid's angle and it's angle velocity.

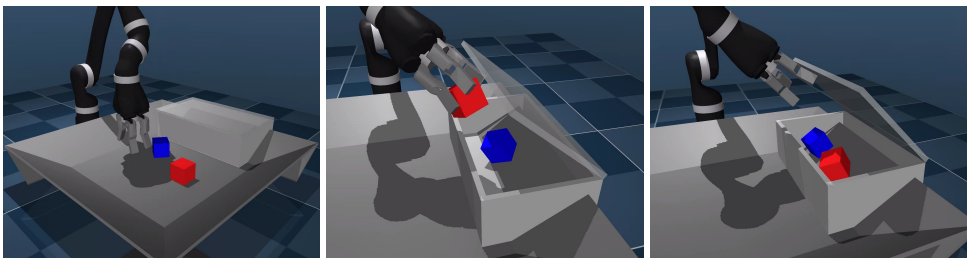


Figure 10: The *Clean-Up* task set-up in simulation. The task is solved when both bricks are in the box.

Analogous to *Pile2* and the SAC-X setup, we use 13 different task for *Clean-Up*. The first 12 tasks are seen as auxiliary tasks, that the agent uses to learn the main task *ALL_INSIDE_BOX*. The tasks used in this experiments are:

- *REACH(R)* = $stol(d(TCP, R), 0.01, 0.25)$:
  Minimize the distance of the TCP to the red cube.
- *REACH(B)* = $stol(d(TCP, B), 0.01, 0.25)$:
  Minimize the distance of the TCP to the blue cube.
- *MOVE(R)* = $slin(|\, linvel(R)\,|, 0, 1)$:
  Move the red cube.
- *MOVE(B)* = $slin(|\, linvel(B)\,|, 0, 1)$:
  Move the blue cube.
- *NO_TOUCH* = $1 - GRASP$
  Sparse binary reward, given when neither of the touch sensors is active.
- *LIFT(R)* = $btol(pos_z(R), 0.05)$
  Increase the z-coordinate of the red cube to more than 5cm relative to the table.
- *LIFT(B)* = $btol(pos_z(B), 0.05)$
  Increase the z-coordinate of the blue cube to more than 5cm relative to the table.
- *OPEN_BOX* = $slin(angle(lid), 0.01, 1.5)$
  Open the lid up to 85 degrees.
- *ABOVE_CLOSE(R, BOX)* = $above(R, BOX) * btol(|d(R, BOX)|, 0.2)$
  Bring the red cube to a position above of and close to the box.
- *ABOVE_CLOSE(B, BOX)* = $above(B, BOX) * btol(|d(B, BOX)|, 0.2)$
  Bring the blue cube to a position above of and close to the box.
- *INSIDE(R, BOX)* = $inside(R, BOX)$
  Place the red cube inside the box.
- *INSIDE(B, BOX)* = $inside(R, BOX)$
  Place the blue cube inside the box.
- *INSIDE(ALL, BOX)* = $INSIDE(R, BOX) * INSIDE(B, BOX)$
  Place the all cubes inside the box.

The sparse reward *inside(A, BOX)* is given by comparing the bounds of the object A and the box. If the bounding box of object A is completely within the box's bounds *inside(A, BOX)* is 1, otherwise *inside(A, BOX)* is 0.

A.5.4 Single Task Results

We investigate the benefits of utilizing hierarchical policies in simulated, single-task environments to flexibly incorporate human domain knowledge. Formally, this scenario corresponds to a single task in the task set: $|I| = 1$. In this setting RHPO recovers a variant of MPO [Abdolmaleki et al., 2018b,a] with hierarchical policies; and we hence compare to MPO with a flat (non-hierarchical) policy as the baseline.

We consider two high-dimensional tasks for continuous control: humanoid-run and humanoid-stand from Tassa et al. [2018]. We compare a single Gaussian policy (i.e. MPO) to training a hierarchical policy, a mixture of Gaussians with three components. To incorporate domain knowledge, we evaluate hierarchical models using two different initialization schemes for the components, i.e, 1- We initialize all the components with zero mean and standard deviation of one (in line with the initialization of the single Gaussian). 2- We distribute the initial means of the three components (with a vector of 0, 1 and -1) such that the high-level controller can select from low action and high action policies immediately. The categorical distribution is initialized as a uniform distribution over components.

In the presented setup learner and actors work asynchronously. To align the update rates of all approaches for a fair comparison, we additionally consider a setting with matching learner speed for single Gaussian and mixture of Gaussians[4]. Figure 11 visualizes the results in terms of both the number of episodes and the gradient steps. As can be observed, the hierarchical policy with distributed means initialization leads to faster learning both in terms of gradient steps and generated data (with approximately aligned update rates). The improved performance in this scenario is clearly connected to the incorporation of domain knowledge which affects the initial policy structure.

[4]We found that optimizing a mixture of Gaussians computationally is more expensive than a single Gaussian resulting in less gradient step w.r.t. data generation rate in an asynchronous setup.

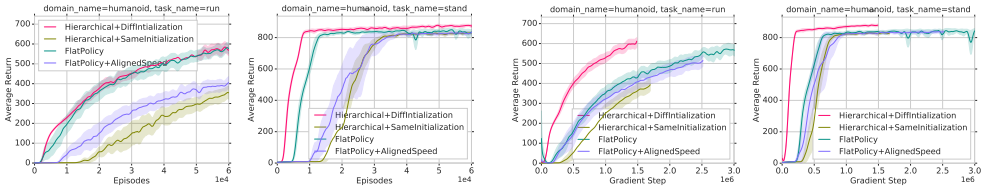


Figure 11: Using a hierarchical policy with different component initialization (red curve) demonstrates benefits over homogeneous initialization as well as the non-hierarchical single Gaussian policy. The plot shows learning curves both in terms of gradient step and number of episodes suggesting that with similar update rate for all models (as in synchronous settings), hierarchy - by enabling flexible incorporation of domain knowledge - gives a considerable advantage in terms of sample efficiency while always faster in terms of learner steps.

A.6 Multitask Results

A.6.1 Pile1 – All Tasks

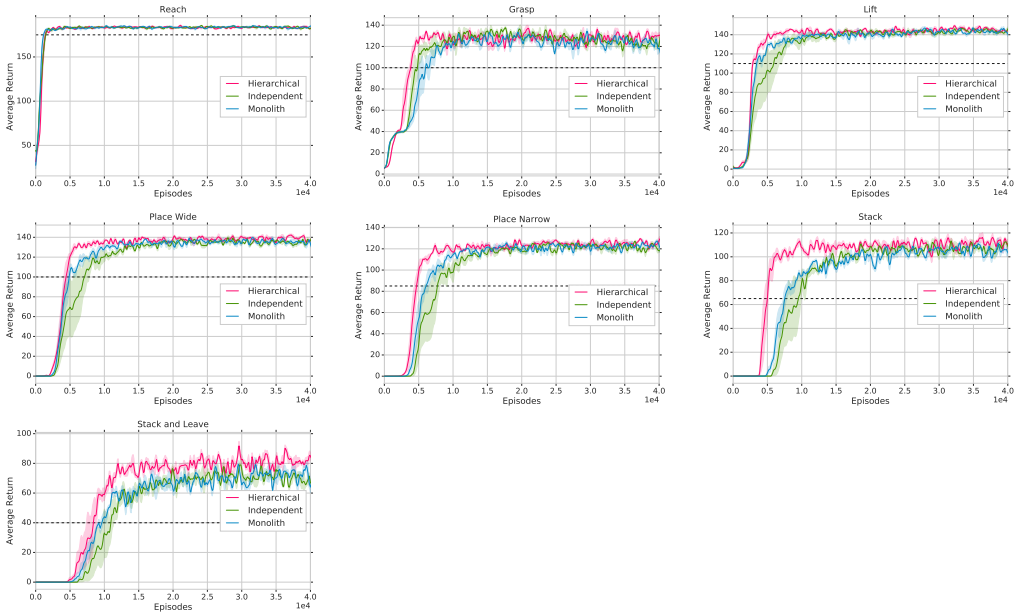


Figure 12: Complete results for all tasks from the Pile1 domain. The dotted line represents standard SAC-U after the same amount of training. Results show that using hierarchical policy leads to best performance.

A.6.2 Pile2 – All Tasks

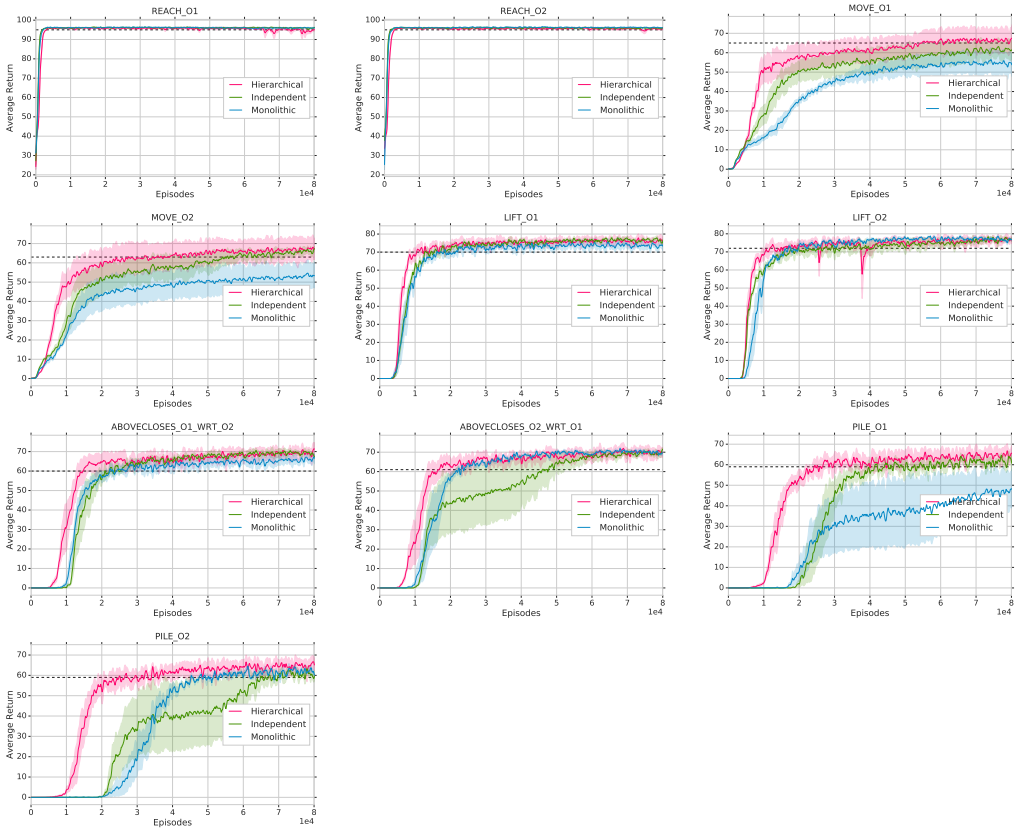


Figure 13: Complete results for all tasks from the Pile2 domain. Results show that using hierarchical policy leads to best performance. The dotted line represents standard SAC-U after the same amount of total training time.

A.6.3 Cleanup2 – All Tasks

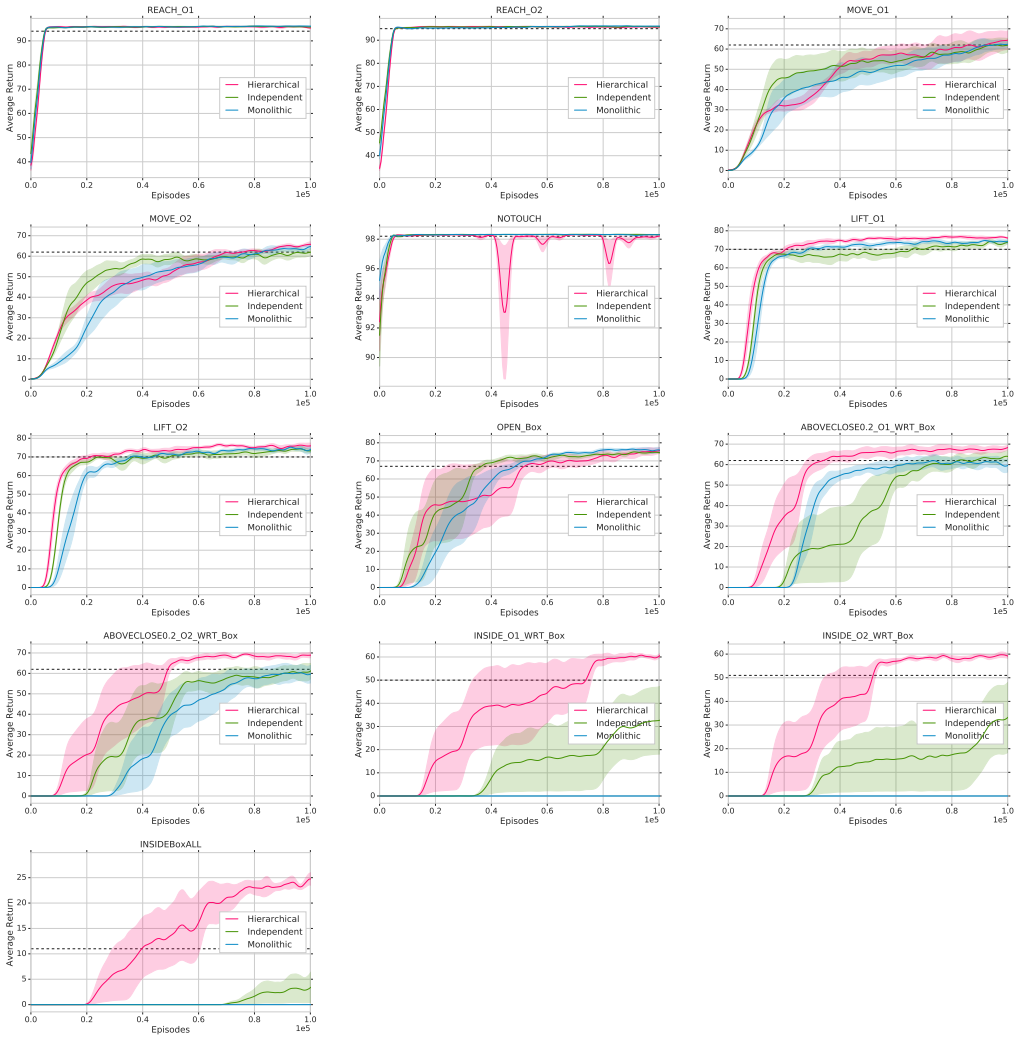


Figure 14: Complete results for all tasks from the Cleanup2 domain. Results show that using hierarchical policy leads to best performance. The dotted line represents standard SAC-U after the same amount of total training time.

A.7 Physical Robot Pile1 – All Tasks

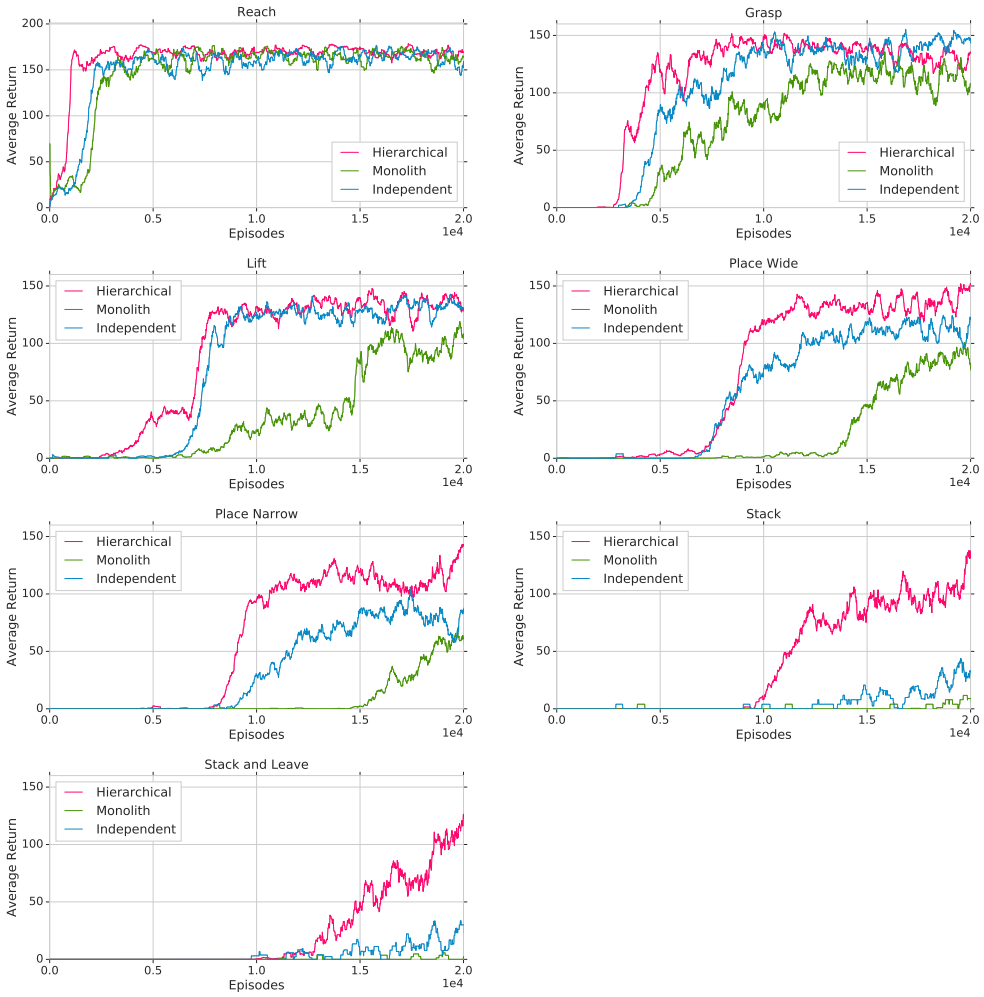


Figure 15: Complete results for all tasks on the real robot Pile1 domain. Results show that using hierarchical policy leads to best performance.

A.8 Additional Ablations Multitask

A.8.1 Number of Component Policies

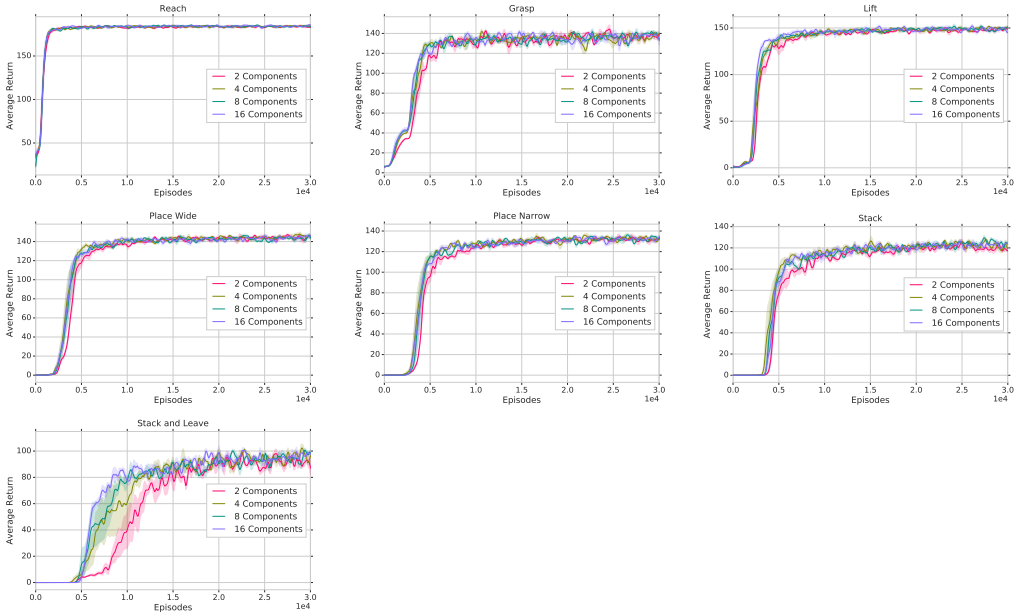


Figure 16: Complete results 2,4,8 and 16 low-level policies in the Pile1 domain. The approach is robust with respect to the number of sub-policies and we will build all further experiments on setting the components equal to the number of tasks.

A.8.2 Kullback Leibler Constraint Ablation

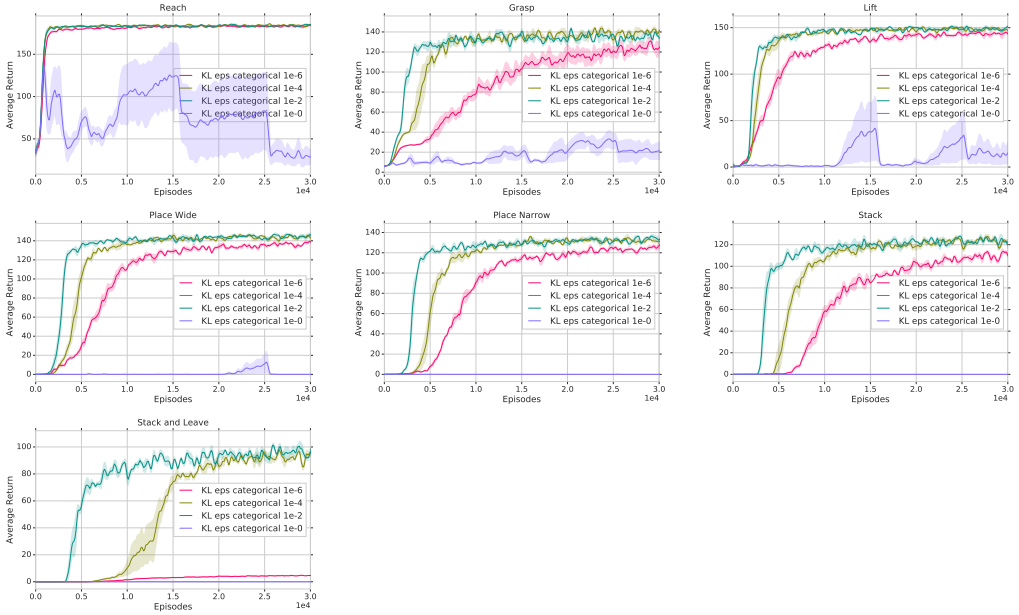
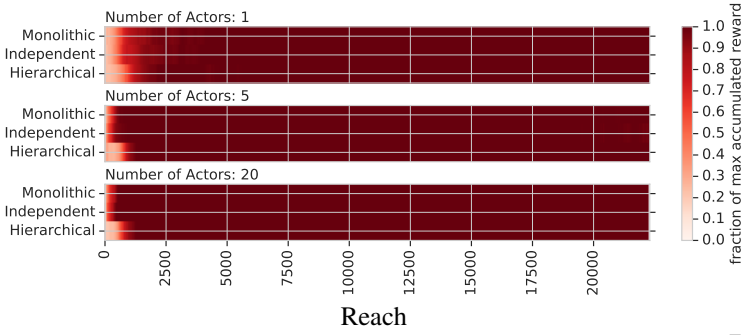
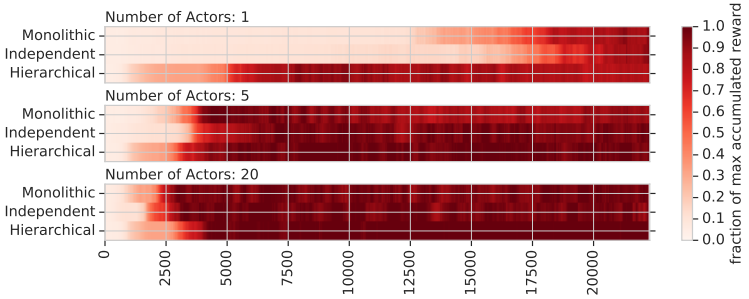


Figure 17: Complete results for sweeping the KL constraint between 1e-6 and 1. in the Pile1 domain. For very weak constraints the model does not converge successfully, while for very strong constraints it only converges very slowly.

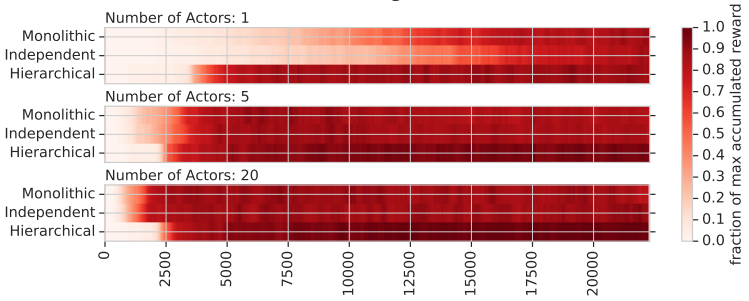
A.8.3 Data Rate Ablation



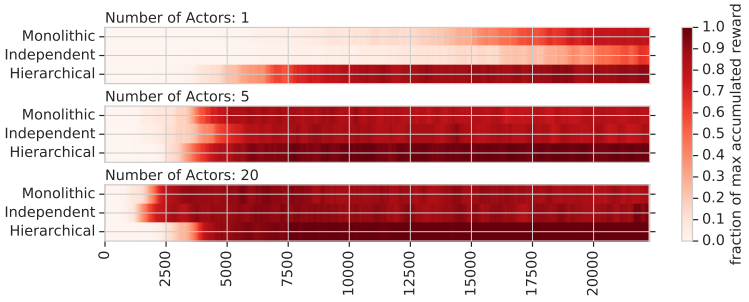
Reach



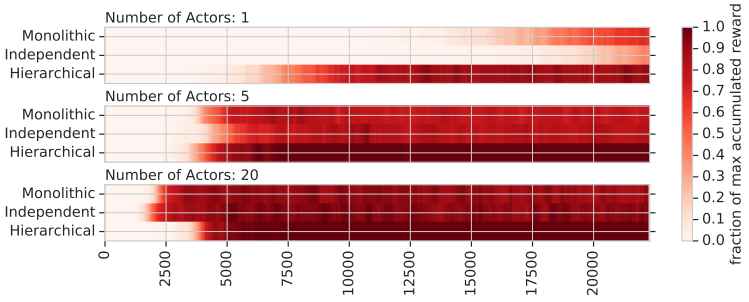
Grasp



Lift



Place Wide



Place Narrow

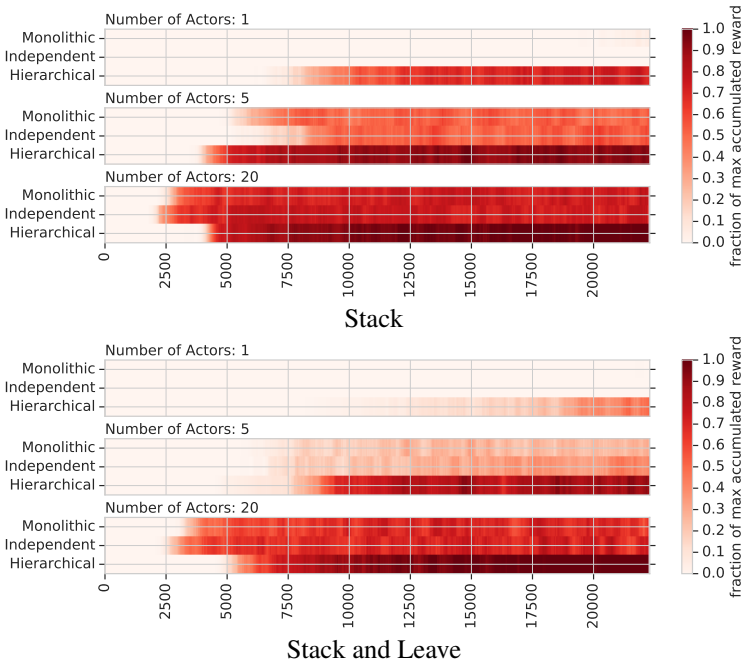


Figure 18: Complete results for ablating the number of data-generating actors in the Pile1 domain. We can see that the benefit of hierarchical policies is stronger for more complex tasks and lower data rates. However, even with 20 actors we see better final performance and stability

A.8.4 Hierarchical Policies in Gradient-based RL

To test whether the benefits of a hierarchical policy transfer to a setting where a different algorithm is used to optimize the policy we performed additional experiments using SVG [Heess et al., 2015] in place of MPO. For this purpose we use the same hierarchical policy structure as for the MPO experiments but change the categorical to an implementation that enables reparameterization with the Gumbel-Softmax trick [Maddison et al., 2016, Jang et al., 2016]. We then change the entropy regularization from Equation (15) to a KL towards a target policy (as entropy regularization did not give stable learning in this setting) and use a regularizer equivalent to the distance function (per component KL's from Equation (12)) – using a multiplier of 0.05 for the regularization multiplier was found to be the best setting via a coarse grid search. This is similar to previous work on hierarchical RL with SVG [Tirumala et al., 2019]. The results of this experiment are depicted in Figure 19, as can be seen, for this simple domain results in mild improvements over standard SAC-U.

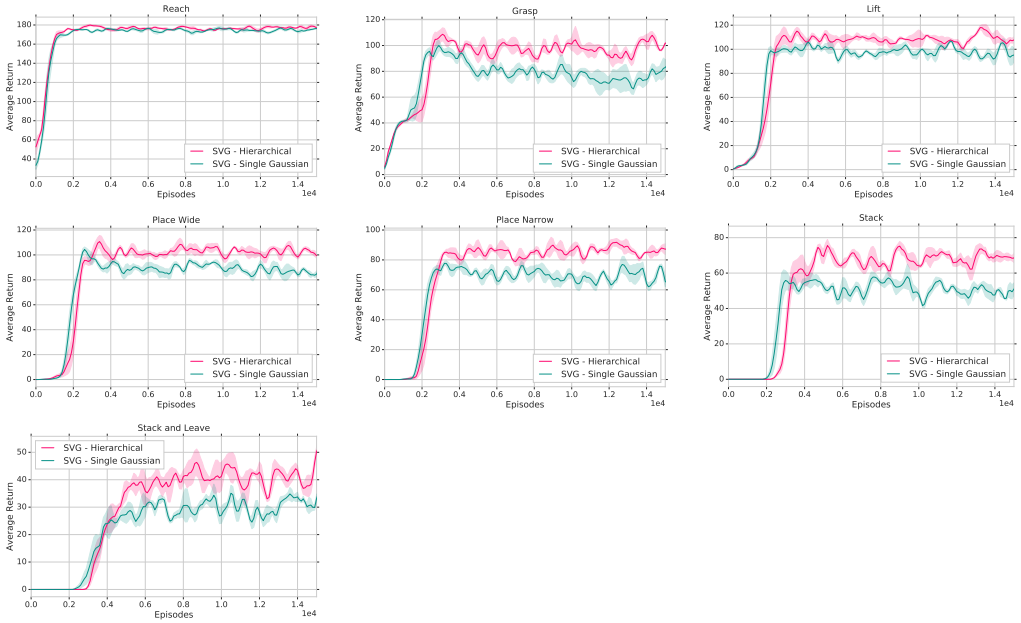


Figure 19: Complete results for evaluating SVG with and without hierarchical policy class in the Pile1 domain. Similarly to the experiments in the main paper, we can see that the hierarchical policy leads to better final performance – here for a gradient-based approach.