Compositional Transfer in Hierarchical Reinforcement Learning

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 general reinforcement learning algorithms to real-world robotics applications is often limited by their high data requirements. We introduce Regularized Hierarchical Policy Optimization (RHPO) to improve data-efficiency for domains with multiple dominant tasks and ultimately reduce required platform time. To this end, we employ compositional inductive biases on multiple levels and corresponding mechanisms for sharing off-policy transition data across low-level controllers and tasks as well as scheduling of tasks. The presented algorithm enables stable and fast learning for complex, real-world domains in the parallel multitask and sequential transfer case. We show that the investigated types of hierarchy enable positive transfer while partially mitigating negative interference and evaluate the benefits of additional incentives for efficient, compositional task solutions in single task domains. Finally, we demonstrate substantial data-efficiency and final performance gains over competitive baselines in a week-long, physical robot stacking experiment.

I. INTRODUCTION

Creating real-world systems that learn to achieve many goals directly through interaction with their environment is one of the long-standing dreams in robotics. Although recent successes in deep (reinforcement) learning for computer games (Atari [28], StarCraft [55]), Go [44] and other simulated environments (e.g. [34]) have demonstrated the potential of these methods when large amounts of training data are available, the high cost of data acquisition has limited progress for many problems involving systems directly acting in the physical world.

Data efficiency in machine learning generally relies on inductive biases or prior knowledge to guide and accelerate the learning process. One strategy for injecting prior knowledge that is widely and successfully used in robotics learning problems is the use of human expert demonstrations to bootstrap the learning process. But the perspective of a system with a permanent embodiment capable of achieving many goals in a persistent environment provides us with a complementary opportunity: an efficient learning strategy should allow us to share and reuse experience across tasks – such that the system does not have to experience or learn the same thing multiple times, and such that solutions to simpler tasks can bootstrap the learning of harder ones.

Rather than providing prior knowledge or biases specific to a particular task this suggests focusing on more general inductive biases that facilitate the sharing and reuse of experience and knowledge across tasks while allowing other aspects

*Correspondence to: mwulfmeier, abdolmaleki@google.com Shared first-authorship.
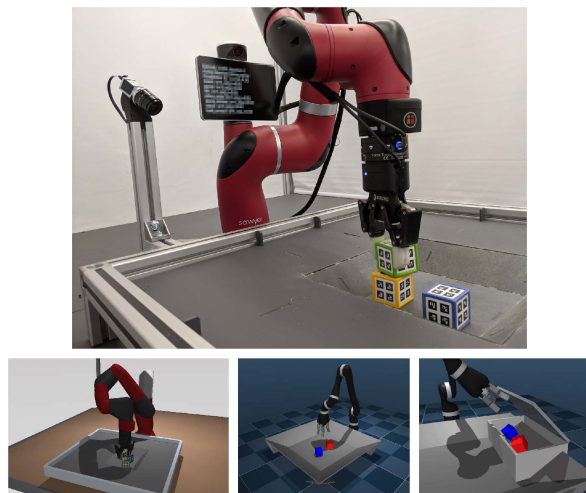

Fig. 1: Top: Overview of the real robot setup with the Sawyer robot performing the Pile1 task. Screen pixelated for anonymization. Bottom Left: Simulated Sawyer performing the same task. Bottom Middle & Right: Respectively Pile2 & Cleanup2 setup with a simulated Jaco arm.

of the domain to be learned [9]. Previous approaches to transfer learning have, for example, built on optimizing initial parameters [e.g. 13], sharing models and parameters across tasks either in the form of policies or value functions [e.g. 41, 51, 15], data-sharing across tasks [e.g. 38, 5], or through the use of task-related auxiliary objectives [23, 57]. Transfer between tasks can, however, lead to either constructive or destructive transfer for humans [45] as well as for machines [35, 53]. That is, jointly learning to solve different tasks can provide both benefits and disadvantages for individual tasks, depending on their similarity. Finding a mechanism that enables transfer where possible but avoids interference is one of the long-standing research challenges.

In this paper, we propose a general reinforcement learning architecture that benefits from learning multiple tasks simultaneously and is sufficiently data-efficient and reliable to solve non-trivial manipulation tasks from scratch directly on robotics hardware. We achieve efficiency through three forms of transfer: (1) robust off-policy learning allows to effectively share all generated transition data across tasks and skills; (2) a modular hierarchical policy architecture allows skills to be directly reused across tasks; and (3) switching between the execution of policies for different tasks within a single episode leads to effective exploration.

The model uses deep neural networks to parameterize state-conditional Gaussian mixture distributions as agent policies, similar to Mixture Density Networks [7]. To obtain robust and versatile low-level behaviors in the multitask setting we shield the mixture components from information about the task at hand. Task information is thus only communicated through the choice of mixture component by the high-level controller, and the mixture components are trained as domain-dependent but task-independent skills. To efficiently optimize hierarchical policies in a multitask setting, we develop robust off-policy learning schemes enabling us to use all transition data to train each low-level controller independent of the actually executed one. We focus on Maximum A-Posteriori Policy Optimization (MPO) [3] but also consider a variant of Stochastic Value Gradients (SVG) [20]. For both algorithms we employ trust-region like constraints at both levels of the hierarchy.

We evaluate the approach on several real and simulated robotics manipulation tasks and demonstrate that it outperforms competitive baselines. In particular, it dramatically improves data efficiency on a challenging real-world robotics manipulation task similar to the one considered in [38]: Our model learns to stack blocks from scratch on a single Sawyer robot arm within about a week at which point it demonstrates up to three times higher performance compared to our baselines. We further perform a number of careful ablations. These highlight, among others, the importance of the hierarchical architecture and the importance of the trust-region like constraints for the stability of the learning scheme. Finally, to gain a better understanding of the role of this type of hierarchy in RL, we compare its benefits in the single task and multitask setting. We find that it shows clear benefits advantages in the multitask setting. However, it can fail to improve performance in the single-task case, where additional incentives are required to encourage component specialization similar to the multitask case. These results shed further light on the interaction of model and domain in RL.

In summary, our contributions are as follows,

- Algorithmic improvements: We propose a new method for robust and efficient off-policy optimization of hierarchical policies. Our approach controls the rate of change at both levels of the hierarchy via trust-region like constraints thus ensuring stable learning. Furthermore, it can use all data to train any given low-level component, independent of the component which generated the transition. This enables data efficient training with experience replay and data sharing across tasks.
- Performance improvements: We evaluate our approach on a range of real and simulated robotic manipulation domains. The results confirm that the algorithm scales to complex tasks and significantly reduces interaction time. Particular benefits arise in more complex task sets and the low-data regime. When learning to stack from scratch on the Sawyer robot arm in a week-long experiment, the approach demonstrates up to three times better performance for the most complex tasks.
- Investigation of benefits, shortcomings and requirements:

We perform a careful analysis and ablation of our algorithm and its properties, highlighting in particular, the impact of individual algorithmic and environment properties, as well was the overall robustness to hyperparameter settings.

II. 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. 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 $p(s)$. 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) $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^\infty \gamma^t r(s_t, a_t)|s_0 = s, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)]$. We define multitask learning over a set of tasks $i \in I$ with common agent embodiment as follows: We assume shared state and action spaces and shared transition dynamics; tasks only differ in their reward function $r_i(s, a)$. We consider task conditional policies $\pi(a|s, i)$ with the overall objective defined as

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

where all actions are drawn according to the policy π conditioned on task i, that is, $a_t \sim \pi(\cdot|s_t, i)$ and we used the following definition of the task-conditional state-action value function (Equation 1).

$$Q^\pi(s, a, i) = \mathbb{E}_\pi \Big[\sum_{t=0}^\infty \gamma^t r_i(s_t, a_t) | a_0 = a,$$
$$s_0 = s, a_t \sim \pi(\cdot|s_t, i), s_{t+1} \sim p(\cdot|s_t, a_t) \Big] \tag{1}$$

III. Method

This section introduces Regularized Hierarchical Policy Optimization (RHPO) which focuses on efficient training of modular policies by sharing data across tasks. We first describe the underlying class of mixture policies, followed by details on the critic-weighted maximum likelihood optimization objective used to update structured hierarchical policies in a multitask, off-policy setting. For efficiency in the multitask case, RHPO extends data-sharing and scheduling mechanisms from Scheduled Auxiliary Control with randomized scheduling (SAC-U) [38].

A. Hierarchical Policies

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

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

where π^H and π^L respectively represent a high-level switching controller (a categorical distribution) and a low-level sub-policy (components of the resulting mixture distribution), and o is the index of the sub-policy. θ denotes the parameters of both π^H and π^L, which we seek to optimize. While the number of components has to be decided externally, RHPO is robust with respect to this parameter (Appendix G3). Note that in the above formulation *only the high-level controller π_H is conditioned on the task information i*. This choice introduces a form of information asymmetry [15, 52, 21] that enables the low-level policies to acquire general, task-independent behaviours. This choice strengthens the decomposition of tasks across domains and prevents degenerate solutions that bypass the high-level controller. Intuitively, these sub-policies can be understood as building reflex-like low-level control loops, which perform domain-dependent but task-independent behaviours and can be modulated by higher cognitive functions with knowledge of the task at hand. Figure 2 illustrates the used hierarchical policy architecture.
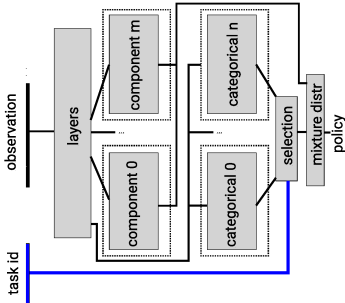

Fig. 2: The hierarchical multitask policy architectures used in this paper. Note that only the high-level controller of mixture distribution is conditioned on the task ID and low level components are shared among tasks. A detailed description can be found in Appendix B3.

B. Data-efficient Multitask Policy Optimization

In the following sections, we present the core principles underlying RHPO; for the complete pseudocode algorithm please see Algorithm 1 and Appendix B1. We build on an Expectation-Maximization based policy optimization algorithm (similar to MPO [2]) and adapt it to the application to hierarchical policies in the multitask case. We update the parametric policy in 2 stages and decouple the policy improvement step from the fitting of the parametric policy.

We begin by describing the policy improvement steps below, 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 Equation (7) for details on learning $\hat{Q}$ from off-policy

data). Starting from an initial policy π_{θ_0} we can then iterate the following steps to improve the policy π_{θ_k}:

leftmargin=*

Policy Evaluation: Update $\hat{Q}$ such that $\hat{Q}(s,a,i) \approx \hat{Q}^{\pi_{\theta_k}}(s,a,i)$, see Equation (7).

Policy Improvement:

– **Step 1:** Obtain $q_k = \arg\max_q J(q)$, under KL constraints with $\pi_{ref} = \pi_{\theta_k}$ (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 (6)).

Policy Improvement 1: Obtaining Non-parametric Policies: Concretely, we first introduce an intermediate non-parametric policy $q(a|s,i)$ and optimize $J(q)$ while staying close, in expectation, to a reference policy $\pi_{ref}(a|s,i)$

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

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.

We find the intermediate policy q by maximizing Equation (3) and can obtain a closed-form solution with a non-parametric policy for each task, as

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

where η is a temperature parameter (corresponding to a given bound ϵ) that is obtained by optimizing the dual function,

$$g(\eta) = \eta\epsilon + \eta\mathbb{E}_{s\sim\mathcal{D},i\sim I}\Big[\log\Big(\int \pi_{\theta_k}(a|s,i)$$
$$\exp\left(\frac{\hat{Q}(s,a,i)}{\eta}\right)\mathrm{d}a\Big)\Big],\qquad(5)$$

(see Appendix A1 for a detailed derivation of the dual function). 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 requirement for obtaining samples. This, crucially, makes it easy to employ complicated structured policies (such as the one introduced in Section III-A). The only requirement here, and in 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 (but the sampling process itself need not be differentiable).

Policy Improvement 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 introduce a trust-region constraint on policy updates. In this way, we can regularize towards a target policy, effectively mitigating

optimization instabilities. Trust-region constraints have been used in on- and off-policy RL [42, 2]. We adapt the formulation of [2] to our hierarchical setting, and as the analysis in Section IV-A shows, it is critical for the success of our algorithm. Formally, we aim to obtain the solution in Equation 6, 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 $\pi_\theta(a|s, i) = \sum_{o=1}^{M} \pi_L(a|s, o) \pi_H(o|s, i)$, highlighting that we are marginalizing over the high-level choices in this fitting step. 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.

$$
\begin{aligned}
\theta_{k+1} &= \arg\min_\theta \mathbb{E}_{s\sim\mathcal{D}, i\sim I}\Big[\mathrm{KL}\big(q_k(\cdot|s, i)\|\pi_\theta(\cdot|s, i)\big)\Big] \\
&= \arg\max_\theta \mathbb{E}_{s\sim\mathcal{D}, i\sim I}\Big[\mathbb{E}_{\pi_{\theta_k}}\Big[\exp(\hat{Q}(s,a,i)/\eta) \\
&\qquad \log \sum_{o=1}^{M} \pi_\theta^L(a|s, o)\, \pi_\theta^H(o|s, i)\Big]\Big], \\
\text{s.t. } & \mathbb{E}_{s\sim\mathcal{D}, i\sim I}\Big[\mathrm{KL}(\pi_{\theta_k}^H(o|s, i)\|\pi_\theta^H(o|s, i))+ \\
&\qquad \frac{1}{M}\sum_{o=1}^{M}\mathrm{KL}(\pi_{\theta_k}^L(a|s, o)\|\pi_\theta^L(a|s, o))\Big] < \epsilon_m
\end{aligned}
\tag{6}
$$

Different approaches can be used to control convergence for both the high-level categorical choices and the action choices to change slowly throughout learning. The average KL constraint in Equation (6) is similar in nature to an 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, 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. To solve Equation (6), 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 [25]); a detailed overview can be found in Algorithm 1 as well as with further information in the Appendix A2.

Policy Evaluation: For data-efficient off-policy learning of $\hat{Q}$ we experience sharing across tasks and switching between tasks within one episode for improved exploration by adapting the initial state distribution of each task based on other tasks [38].

Formally, we assume access to a replay buffer containing data gathered from all tasks. 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

---

Algorithm 1 RHPO - Asynchronous Learner

Input: N_{steps} number of update steps, $N_{\mathrm{targetUpdate}}$ 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 $k \leq N_{\mathrm{steps}}$ **do**
  for k in $[0...N_{\mathrm{targetUpdate}}]$ **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
    $\delta_\pi \leftarrow -\nabla_\theta \sum_{s_t\in\tau}\sum_{j=1}^{N_s}[\exp\big(\frac{Q(s_t,a_j,i)}{\eta}\big)$
              $\log\pi_\theta(a_j|s_t,i)]$ following Eq. 6
    $\delta_\eta \leftarrow \nabla_\eta g(\eta) = \nabla_\eta \eta\epsilon + \eta\sum_{s_t\in\tau}\log\frac{1}{N_s}\sum_{j=1}^{N_s}[$
              $\exp\big(\frac{Q(s_t,a_j,i)}{\eta}\big)]$ following Eq. 5
    $\delta_Q \leftarrow \nabla_\phi \sum_{i\sim I}\sum_{(s_t,a_t)\in\tau}\big(\hat{Q}_\phi(s_t,a_t,i)-Q^{\mathrm{ret}}\big)^2$
              with Q^{ret} following Eq. 7
    // 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

---

learning $\hat{Q}$, parameterized via ϕ, following [31, 38] as

$$
\begin{aligned}
\min_\phi L(\phi) = \sum_{i\sim I}\mathbb{E}_{\tau\sim\mathcal{D}}\Big[\big(r_i(s_t, a_t)+ \\
\gamma Q^{ret}(s_{t+1}, a_{t+1}, i) - \hat{Q}_\phi(s_t, a_t, i))^2\Big],
\end{aligned}
\tag{7}
$$

where Q^{ret} is the L-step retrace target [31], see the Appendix B2 for details.

IV. EXPERIMENTS

In the following sections, we investigate the effects of training hierarchical policies in single and multitask domains. In particular, we demonstrate that RHPO can provide compelling benefits for multitask learning in real and simulated robotic manipulation tasks and significantly reduce platform interaction time. For the final experiment, a stacking task on a physical Sawyer robot arm, RHPO achieves a dramatic performance improvement after a week of training compared to several strong baselines. We further investigate RHPO in a sequential transfer setting and find that when pre-trained skills (i.e. low-level components) are available RHPO can provide additional improvements in data efficiency.

Finally, we perform a number of ablations to emphasize the importance of trust-region constraints for the high-level controller and to understand the relative role of hierarchy

in the single-task and multitask setting: In the single-task case, using domains from the DeepMind Control Suite [49], we first demonstrate that our hierarchy on its own can fail to improve performance and that for the model to exploit compositionality in this setting, additional incentives for component specialization are required.

For all tasks and algorithms, we use a distributed actor-critic framework (similar to [12]) with flexible hardware assignment [8]. We perform critic and policy updates from a replay buffer, which is asynchronously filled by a set of actors. In all figures with error bars, we visualize mean and variance derived from 3 runs. Additional details of task hyperparameters as well as results for ablations and the full set of tasks from the multitask domains are provided in the Appendix D. *

A. Simulated Multitask Experiments

We use three simulated multitask scenarios with the Kinova Jaco and Rethink Robotics Sawyer robot arms to test in a variety of conditions. The three scenarios each consist of tasks of different difficulties and vary in their overall complexity. The least difficult scenario is **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. The two more difficult scenarios are **Pile2** and **Cleanup2**. **Pile2** includes stacking with both blocks on top of the respective other block, resulting in a setting with 10 tasks. **Cleanup2** includes harder tasks such as opening a box and placing blocks into this box, consisting of a total of 13 tasks. In addition to the experiments in simulation, which are executed with 5 actors in a distributed setting, we also investigate the **Pile1** multitask domain (same rewards and setup) on a single, physical robot in Section IV-B.

Our main comparison evaluates **RHPO** with hierarchical policies against SAC [38] with a flat, monolithic policy shared across all tasks which is provided with the additional task id as input (displayed as **SAC-U-Monolithic**) as well as policies with task dependent heads (displayed as **SAC-U-Independent**) following [38] – both using MPO as the optimizer. Furthermore, we compare against a re-implementation of SAC using SVG [20] as actor-critic based optimizer which uses the reparameterization trick (displayed as SAC-U[SVG]). In order to compare with gradient-based hierarchical policy updates (such option critic [6]) as well as investigating the application of the proposed hierarchical model for other RL algorithms; we also use SVG (with continuous relaxation of the Categorical distribution [27, 24]) instead of MPO to optimize the hierarchical model with results included in the Pile1 experiments (displayed as RHPO[SVG]) with additional results in Appendix H. These comparisons furthermore strengthen our choice for critic-weighted likelihood instead of reparametrization gradient-based policy optimizer.

The main SAC baselines provide the two opposite, naive perspectives on transfer: by using the same monolithic policy across tasks we enable positive as well as negative interference

*Additional details and the appendix can be found under https://sites. google.com/corp/view/rhpo

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 G3), we set M proportional to the number of tasks in each domain.
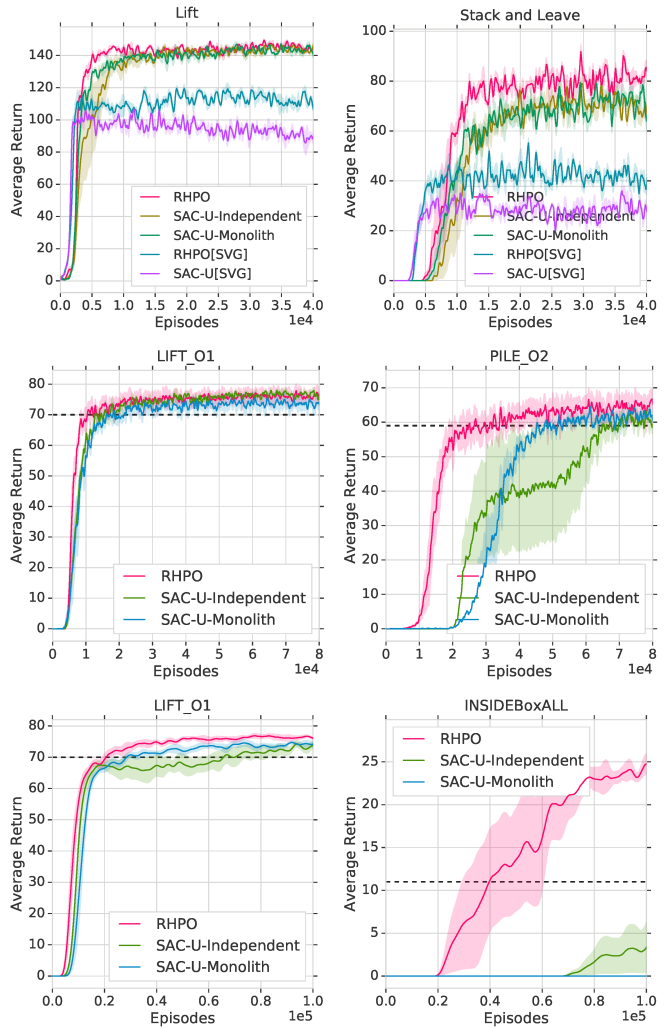

Fig. 3: Results for the multitask robotic manipulation experiments in simulation. The dashed line corresponds to the performance of the SVG-based implementation of SAC-U. From top to bottom: 2 tasks from the Pile1, Pile2 & Cleanup2 domains. We show averages over 3 runs each, with corresponding standard deviation. RHPO outperforms both baselines across all tasks with the benefits increasing for more complex domains.

Figure 3 demonstrates that RHPO outperforms the monolithic as well as the independent baselines (based on SAC). For simple tasks such as the Pile1 domain, the difference is small, but as the number of tasks grows and the complexity of the domain increases (cf. Pile2 and Cleanup2), the advantage of composing learned behaviours across tasks becomes more significant. We further observe that using MPO instead of SVG [20] as policy optimizer results in an improvement for the baselines. This effect becomes more pronounced for the hierarchical policies.
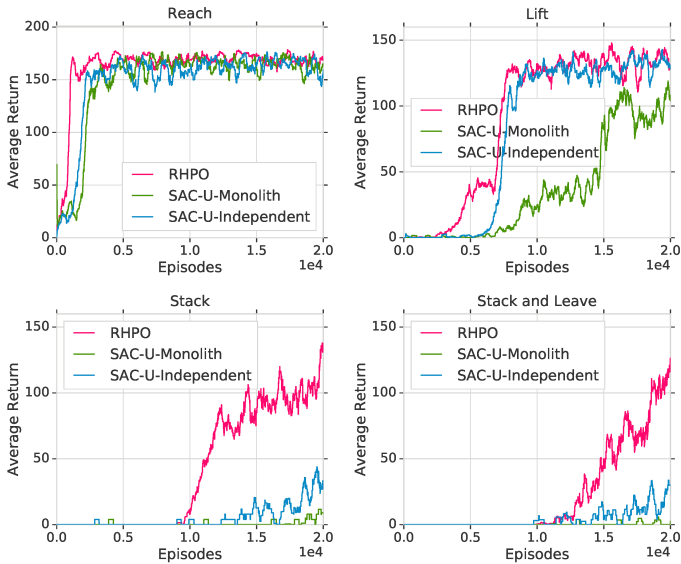
Fig. 4: Robot Experiments. Left: While simpler tasks such as reaching are learned with comparable efficiency, the later, more complex tasks are acquired significantly faster with RHPO.

B. Physical Robot Experiments

For real-world experiments, data-efficiency is crucial. We perform all experiments in this section relying on a single robot (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 IV-A. Given the higher cost of experiment time, the robot experiments additionally emphasize the requirements for hyperparameter robust algorithms which is further investigated in Section IV-E.

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's 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 4 plots the learning progress on the real robot for four of the tasks, from simpler reach and lift tasks and the stack and final stack-and-leave task – which is the main task of interest. Plots for the learning progress of all tasks are given in the appendix F. 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 considerably slower progress for all baselines taking multiple weeks for completion.

To provide further insight into the learned representation we compute distributions for each component over the tasks which activate it, as well as distributions for each task over which components are being used. For each set of distributions, we determine the Battacharyya distance metric to determine the similarity between tasks and the similarity between components in Figure 5 (right). The plots demonstrate how the components specialize, but also provide a way to investigate our tasks, showing e.g. that the first reach task is fairly independent and that the last four tasks are comparably similar regarding the high-level components applied for their solution.

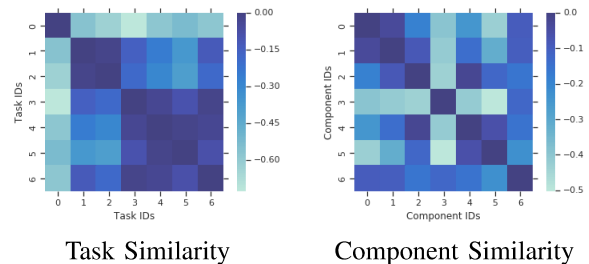
Task Similarity   Component Similarity

Fig. 5: Similarities between tasks (based on their distribution over components) and similarities between components (based on the distribution over tasks which apply them).
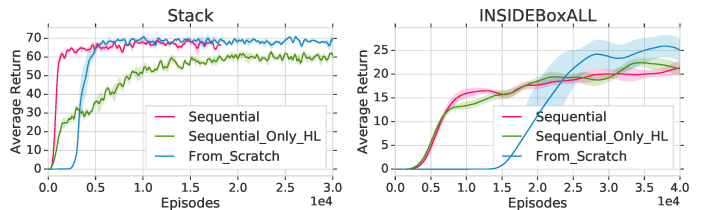
C. Sequential Transfer

Fig. 6: Sequential transfer experiments: the models are first trained with all but the final task in the Pile1 and Cleanup2 domains, and finally we train the models to adapt to the final task by either training 1- only a high-level controller or 2-a high-level controller as well as an additional component.

RHPO is well suited for sequential transfer learning as it allows to use pre-trained low-level components to solve new tasks. To investigate performance in adapting pre-trained multitask policies to novel tasks, we train agents to fulfill all but the final task in the Pile1 and Cleanup2 domains and subsequently evaluate training the models on the final task. We consider two settings for the final policy: we either introduce only a new high-level controller (Sequential-Only-HL) or both an additional shared component as well as a new high-level controller (Sequential). Figure 6 displays that in the sequential transfer setting, starting from a policy trained on a set of related tasks results in up to 5 times more data-efficiency in terms of actor episodes on the final task than training the same policy from scratch. We observe that the final task can be solved by only reusing low-level components from previous tasks if the final task is the composition of previous tasks. This is the case for the final task in Cleanup2 which can be completed by sequencing the previously learned components and in contrast

to Pile1 where the final letting go of the block after stacking is not required for earlier tasks.

D. Simulated Single Task Experiments

We consider two high-dimensional tasks for continuous control: humanoid-run and humanoid-stand from Tassa et al. [49] and compare MPO with a flat Gaussian policy to RHPO with a mixture of Gaussians with three components. Figure 7 shows the results in terms of the number of episodes.

When both the flat and hierarchical policies are initialized with means close to zero, RHPO performs comparable to a flat policy and learns similar means and variances for all components as the model fails to decompose the learned behavior. If, however, the hierarchical policy is initialized with distinct means for different components (here, for the three components ranging for all dimensions from the minimum to maximum of the allowed action range, i.e. -1, 1), we observe significantly improved performance and component specialization.
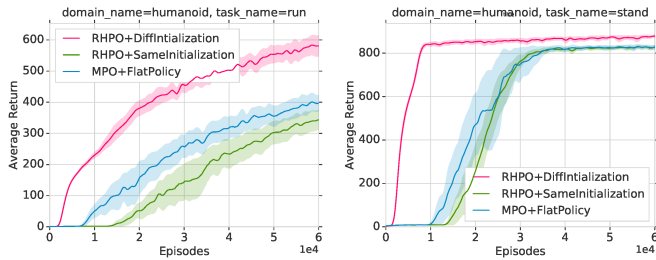

Fig. 7: Using RHPO with different component initialization (red curve) demonstrates benefits over homogeneous initialization as well as the flat Gaussian policy. The plot shows that the simple change in initialization is sufficient to enable component specialization and the correlated improvement in performance.

E. Performance Ablations

We perform a series of ablations based on the earlier introduced Pile1 domain, providing additional insights into benefits, shortcomings and relevant hyperparameters of RHPO.

First, we display the importance of choice of regularization in Figure 8 with complete results in Appendix G1. We are able to demonstrate the effect of weakening the constraint by setting the epsilon value higher (here: to 1.). This setting prevents convergence of the policy to capable solutions and emphasizes the necessity of constraining the update steps. In addition, very small values can slow down convergence. However, in the present experiments a range of about 2 orders of magnitude results in good performance.

We additionally ablate over the number of data-generating actors in Figure 9 to evaluate all approaches with respect to data rate and illustrate how RHPO is particularly relevant at lower data rates such as given by real-world robotics applications (with results for all tasks in Appendix G2). Here, RHPO always provides stronger final performance and learns significantly faster in one actor case as common for robot experiments. By running with multiple actors, we increase the rate of data
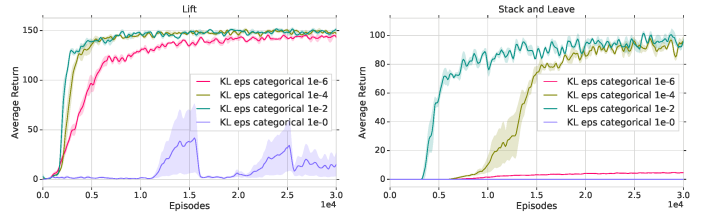

Fig. 8: Results for sweeping the KL constraint between 1e-6 and 1. for 2 tasks in the Pile1 domain. For very weak constraints the model does not converge successfully, while for very strong constraints it only converges very slowly.

generation such that in asynchronous settings, the speed of the learner becomes more important. Since training our hierarchical policies is computationally slightly more costly, the benefits become smaller for easier tasks[†].
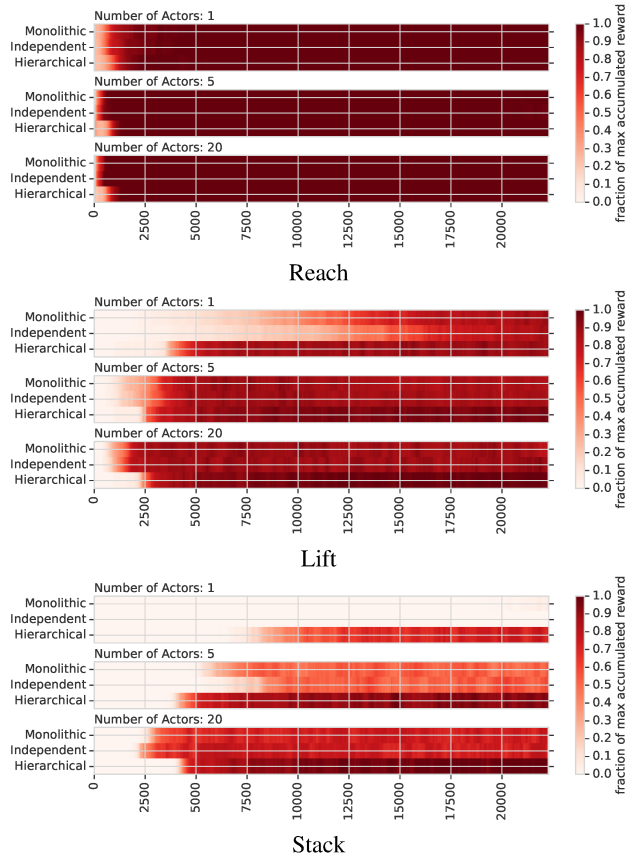

Fig. 9: Results for ablating the number of data-generating actors in the Pile1 domain. We can see that the benefit of hierarchical policies is stronger for more complex tasks and lower data rates. However, even with 20 actors we see better final performance and stability

Finally, we demonstrate the robustness of RHPO with respect to the number of sub-policies in Figure 9 (with complete results in Appendix G3) and connected simplicity of determining this hyperparameter which for all other experiments is simply set proportionally to the number of tasks.

[†]In asynchronous RL systems, the update rate of the learner can have a significant impact on the performance when evaluated over data generated.

V. Related Work

Transfer learning, in particular in the multitask context, has long been part of machine learning (ML) for data-limited domains [9, 53, 35, 50]. 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 [56]. Preventing this type of forgetting or negative transfer presents a challenge for biological [45] as well as artificial systems [14]. In the context of ML, a common scheme is the reduction of representational overlap [14, 41, 56]. Bishop [7] utilize neural networks to parametrize mixture models for representing multi-modal distributions thus mitigating shortcomings of non-hierarchical approaches. Rosenstein et al. [40] demonstrate the benefits of hierarchical classification models to limit the impact of negative transfer.

Hierarchical approaches have a long history in the reinforcement learning literature [e.g. 48, 11]. Prior work commonly benefits from combining hierarchy with additional inductive biases such as [54, 33, 32, 58] 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 [43, 4, 52]. Instead of introducing additional training signals, we directly investigate the benefits of compositional hierarchy as provided structure for transfer between tasks.

Hierarchical models for probabilistic trajectory modelling have been used for the discovery of behavior abstractions as part of an end-to-end reinforcement learning paradigm [e.g. 51, 22, 52, 15] where the models act as learned inductive biases that induce the sharing of behavior across tasks. In a vein similar to the presented algorithm, [e.g 21, 52] 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 [19, 16] learn hierarchical policies with continuous latent variables optimizing the entropy regularized objective.

Similar to our work, the options framework [48, 36] supports behavior hierarchies, 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. 6, 59, 46, 39, 17] or criteria for option induction [e.g. 17, 18]. Rather than the additional inductive bias of temporal abstraction, we focus on the investigation of composition as type of hierarchy in the context of single and multitask learning while demonstrating the strength of hierarchical composition to lie in domains with strong variation in the objectives - such as in multitask domains. We additionally introduce a hierarchical extension of SVG [20], to investigate similarities to work on the option critic [6].

With the use of KL regularization to different ends in RL, work related to RHPO focuses on contextual bandits [10]. The algorithm builds on a 2-step EM like procedure to optimize linearly parametrized mixture policies. However, their algorithm has been used only with low dimensional policy representations, and in contextual bandit and other very short horizon settings. Our approach is designed to be applicable to full RL problems in complex domains with long horizons and with high-capacity function approximators such as neural networks. This requires robust estimation of value function approximations, off-policy correction, and additional regularization for stable learning.

VI. Discussion

We introduce RHPO, a novel algorithm for robust training of hierarchical policies in multitask settings. RHPO consistently outperforms competitive baselines which either handle tasks independently or implicitly share experience by reusing data across tasks. Especially for complex tasks or in a low data regime, as encountered in robotics applications, we strongly reduce the number of environment interactions and improve final performance as well as learning robustness and sensitivity to hyper-parameters. Our results show that the algorithm scales to complex, real-world domains and provides an important step towards the deployment of RL algorithms on robotic systems.

Algorithmically, our method highlights the importance of trust-region-like regularization for stable optimization of hierarchical policies. Furthermore, our update rules in combination with mixture policies and hindsight reward assignments enable training for any task and skill independent of the data source. This enables efficient learning of the hierarchical policies in an off-policy setting, which is important for data efficient learning.

Conceptually, our results demonstrate that hierarchical policies can be an effective way of sharing skills or behavior components across tasks, both in multitask (Sections IV-A-IV-B) as well as in transfer settings (Section IV-C) and partially mitigate negative interference between tasks in the parallel multitask learning scenario. Furthermore, we find that their benefits are complementary to off-policy sharing of transition data across tasks (e.g. SAC-X [38], HER [5]). Valuable directions for future work include the direct extension to multilevel hierarchies and the identification of basis sets of behaviours which perform well on wide ranges of possible tasks given a known domain.

We believe that especially in domains with consistent agent embodiment and high costs for data generation learning tasks jointly and information sharing is imperative. Our results suggest that a system that is exposed to a rich set of tasks or experiences and has appropriate means for reusing knowledge can learn to solve non-trivial problems directly from interaction with its environment. RHPO combines several ideas that we believe will be important: sharing data across tasks and skills across tasks with compositional 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, more importantly, on physical hardware.

REFERENCES

[1] 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.

[2] 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*, 2018.

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

[4] 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.

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

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

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

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

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

[27] 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.

[28] 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.

[29] 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.

[30] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.

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

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

[33] 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, 2018.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. In *Advances in Neural Information Processing Systems*, pages 10424–10434, 2018.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

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

[46] 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.

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

[48] 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.

[49] 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.

[50] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

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

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

[58] 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.

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