Multitask Soft Option Learning

Maximilian Igl[*] University of Oxford
Andrew Gambardella University of Oxford
Jinke He Delft University of Technology
Nantas Nardelli University of Oxford
N. Siddharth University of Oxford
Wendelin Böhmer University of Oxford
Shimon Whiteson University of Oxford

Abstract

We present Multitask Soft Option Learning (MSOL), a hierarchical multitask framework based on Planning as Inference. MSOL extends the concept of options, using separate variational posteriors for each task, regularized by a shared prior. This "soft" version of options avoids several instabilities during training in a multitask setting, and provides a natural way to learn both intra-option policies and their terminations. Furthermore, it allows fine-tuning of options for new tasks without forgetting their learned policies, leading to faster training without reducing the expressiveness of the hierarchical policy. We demonstrate empirically that MSOL significantly outperforms both hierarchical and flat transfer-learning baselines.

1 INTRODUCTION

A key challenge in Deep Reinforcement Learning is to scale current approaches to complex tasks without requiring a prohibitive number of environmental interactions. One promising approach is to construct or learn efficient exploration priors to focus on more relevant parts of the state-action space, reducing the number of required interactions. This includes, for example, reward shaping (Ng et al., 1999), curriculum learning (Bengio et al., 2009), meta-learning (Wang et al., 2016) and transfer learning (Teh et al., 2017).

In particular, transfer learning does not require human designed rewards or curricula, instead allowing the network to learn what and how to transfer knowledge between tasks. One promising way to capture such knowledge is to decompose policies into a hierarchy of sub-policies (or skills) that can be reused and combined in novel ways to solve new tasks (Sutton et al., 1999). This idea of Hierarchical RL (HRL) is also supported by findings that humans appear to employ a hierarchical mental structure when solving tasks (Botvinick et al., 2009). In such a hierarchical policy, lower-level, *temporally extended* skills yield directed behavior over multiple time steps. This has two advantages: i) it allows efficient exploration, as the target states of skills can be reached without having to explore much of the state space in between, and ii) directed behavior also reduces the variance of the future reward, which accelerates convergence of estimates thereof. On the other hand, while a hierarchical approach can significantly speed up exploration and training, it can also severely limit the expressiveness of the final policy and lead to suboptimal performance when the temporally extended skills are not able to express the required policy for the task at hand.

Many methods exist for learning such hierarchical skills, (e.g. Sutton et al., 1999; Bacon et al., 2017; Gregor et al., 2016). The key challenge is to learn skills which are diverse, and relevant for future tasks. One widely used approach is to rely on additional human-designed input, often in the form of manually specified subgoals (Vezhnevets et al., 2017; Nachum et al., 2018) or a fixed temporal extension of learned skills (Frans et al., 2018). While this can lead to impressive results, it is only applicable in situations where relevant subgoals or temporal extension can be easily identified a priori.

This paper proposes Multitask Soft Option Learning (MSOL), an algorithm to learn hierarchical skills from a given distribution of tasks without any additional human specified knowledge. MSOL trains simultaneously on multiple tasks from this distribution and autonomously extracts sub-policies which are reusable across them.

Importantly, unlike prior work (Frans et al., 2018), our proposed *soft option* framework avoids several pitfalls of learning options from multiple tasks, which arise when skills are jointly optimized with a higher-level policy that

[*]Corresponding author: maximilian.igl@gmail.com

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

determines when each skill is used. Generally, as each skill must be used for similar purposes across all tasks, to learn consistent behavior, a complex training schedules is required to assure a nearly converged higher-level policy before skills can be updated (Frans et al., 2018). However, once a skill has converged it can be hard to change its behavior without hurting the performance of higher-level policies that rely it. Training is therefore prone to end up in local optima: even if changing a skill on *one* task could increase the return, it would likely lead to lower returns on *other* tasks in which it is currently used. This is particularly an issue when multiple skills have learned similar behavior, preventing the learning of a diverse set of skills.

MSOL alleviates both difficulties. The core idea is to learn a "prototypical" – or *prior* – behavior for each skill, while allowing the actually-executed skill on each task – the *posterior* – to deviate from it if the specific task rewards require it. Penalizing deviations between the prior and posteriors from different tasks gives rise to skills that are consistent across tasks, and can be elegantly formulated in the Planning as Inference (PAI) framework (Levine, 2018). This distinction between prior and task-dependent posterior obviates the need for complex training schedules: every task can change their posterior independently of each other and discover new skills without direct interference in other tasks. Nevertheless, the penalization term encourages skills to be similar across tasks *and* rewards higher-level policies for preferring such more specialised skills. We discuss in more detail in Section 3.5 how this helps to prevent the aforementioned local optima.

In addition to these optimization pitfalls, the idea of soft options also alleviates the restrictiveness of hierarchical policies. New tasks can make use of learned skills, by initializing their posterior skills from the priors, but are not restricted by them. The penalization term between prior and posterior acts here as learned shaping reward, guiding the exploration on new tasks towards previously relevant behavior, without requiring the new policy to exactly match previous behavior. In difference to prior work, MSOL can thus even learn tasks that are not solvable with previously learned skills alone. Finally, we show how the *soft option* framework gives rise to a natural solution to the challenging task of learning option-termination policies.

Our experiments demonstrate that MSOL outperforms previous hierarchical and transfer-learning algorithms during transfer tasks in a multitask setting. Unlike prior work, MSOL only modifies the regularized reward and loss function. and does not require specialized architectures, or artificial restrictions on the expressiveness of either the higher-level or intra-option policies.

2 PRELIMINARIES

An agent's task is formalized as a MDP $(\mathcal{S}, \mathcal{A}, \rho, P, r, \gamma)$, consisting of the state space $\mathcal{S}$, action space $\mathcal{A}$, initial state distribution ρ, transition probability $P(s_{t+1}|s_t, a_t)$ of reaching state s_{t+1} by executing action a_t in state s_t, reward $r(s_t, a_t) \in \mathbb{R}$ that an agent receives for this transition, and discount factor $\gamma \in [0, 1]$. An optimal agent chooses actions that maximize the return $R_t(s_t) = \sum_k \gamma^k r_{t+k}$ consisting of discounted future rewards.

2.1 PLANNING AS INFERENCE

Planning as inference (PAI) (Todorov, 2008; Levine, 2018) frames Reinforcement Learning (RL) as a probabilistic inference problem. The agent learns a distribution $q_\phi(a|s)$ over actions a given states s, i.e., a policy, parameterized by ϕ, which induces a distribution over trajectories τ of length T, i.e., $\tau = (s_1, a_1, s_2, \ldots, a_T, s_{T+1})$:

$$q_\phi(\tau) = \rho(s_1) \prod_{t=1}^{T} q_\phi(a_t|s_t)\, P(s_{t+1}|s_t, a_t)\,. \tag{1}$$

This can be seen as a structured variational approximation of the optimal trajectory distribution. Note that the true initial state probability $\rho(s_1)$ and transition probability $P(s_{t+1}|s_t, a_t)$ are used in the variational posterior, as we can only control the policy, not the environment.

A significant advantage of this formulation is that it is straightforward to incorporate information both from prior knowledge, in the form of a prior policy distribution, and the task at hand through a likelihood function that is defined in terms of the achieved reward. The prior policy $p(a_t|s_t)$ can be specified by hand or, as in our case, learned (see Section 3). To incorporate the reward, we introduce a binary *optimality variable* $\mathcal{O}_t$ (Levine, 2018), whose likelihood is highest along the optimal trajectory that maximizes return: $p(\mathcal{O}_t = 1|s_t, a_t) = \exp\big(r(s_t, a_t)/\beta\big)$, where for $\beta \to 0$ we recover the original RL problem. The constraint $r \in (-\infty, 0]$ can be relaxed without changing the inference procedure (Levine, 2018). For brevity, we denote $\mathcal{O}_t = 1$ as $\mathcal{O}_t \equiv (\mathcal{O}_t = 1)$. If a given prior policy $p(a_t|s_t)$ explores the state-action space sufficiently, then $p(\tau, \mathcal{O}_{1:T})$ is the distribution of desirable trajectories. PAI aims to find a policy such that the variational posterior in (1) approximates this distribution by minimizing the Kullback-Leibler (KL) divergence:

$$\begin{aligned} \mathcal{L}(\phi) &= \mathbb{D}_{\text{KL}}(q_\phi(\tau) \,\|\, p(\tau, \mathcal{O}_{1:T}))\,, \text{ where} \\ p(\tau, \mathcal{O}_{1:T}) &= \rho(s_1) \prod_{t=1}^{T} p(a_t|s_t) P(s_{t+1}|s_t, a_t) p(\mathcal{O}_t|s_t, a_t). \end{aligned} \tag{2}$$

2.2 MULTI-TASK LEARNING

In a multi-task setting, we have a set of different tasks $i \in \mathcal{T}$, drawn from a task distribution with probability $\xi(i)$. All tasks share state space $\mathcal{S}$ and action space $\mathcal{A}$, but each task has its own initial-state distribution ρ_i, transition probability $P_i(s_{t+1}|s_t, a_t)$, and reward function r_i. Our goal is to learn n tasks concurrently, distilling common information that can be leveraged to learn faster on new tasks from $\mathcal{T}$. In this setting, the prior policy $p_\theta(a_t|s_t)$ can be learned jointly with the task-specific posterior policies $q_{\phi_i}(a_t|s_t)$ (Teh et al., 2017). To do so, we simply extend (2) to

$$\begin{aligned}\mathcal{L}(\{\phi_i\}, \theta) &= \mathbb{E}_{i\sim\xi}\left[\mathbb{D}_{\text{KL}}(q_{\phi_i}(\tau) \,\|\, p_\theta(\tau, \mathcal{O}_{1:T}))\right] \\ &= -\frac{1}{\beta}\,\mathbb{E}_{i\sim\xi,\tau\sim q}\left[\sum_{t=1}^{T} R^{\text{reg}}_{i,t}\right], \end{aligned} \tag{3}$$

where $R^{\text{reg}}_{i,t} := r_i(s_t, a_t) - \beta \ln \frac{q_{\phi_i}(a_t|s_t)}{p_\theta(a_t|s_t)}$ is a regularised reward. Minimizing the loss in (3) is equivalent to maximizing the regularized reward $R^{\text{reg}}_{i,t}$. Moreover, minimizing the term $\mathbb{E}_{\tau\sim q}\left[\ln \frac{q_{\phi_i}(a_t|s_t)}{p_\theta(a_t|s_t)}\right]$ implicitly minimizes the expected KL-divergence $\mathbb{E}_{s_t\sim q}\left[\mathbb{D}_{\text{KL}}[q_{\phi_i}(\cdot|s_t)\|p_\theta(\cdot|s_t)]\right]$. In practise (see Appendix B.1) we will also make use of a discount factor $\gamma \in [0, 1]$. For details on how γ arises in the PAI framework we refer to Levine (2018).

2.3 OPTIONS

Options (Sutton et al., 1999) are skills that generalize primitive actions and consist of three components: i) an intra-option policy $p(a_t|s_t, z_t)$ that selects primitive actions according to the currently active option z_t, ii) a probability $p(b_t|s_t, z_{t-1})$ of terminating the *previously* active option z_{t-1}, and iii) an initiation set $\mathcal{I} \subseteq \mathcal{S}$, which we simply assume to be $\mathcal{S}$. Note that by construction, the higher-level (or master-) policy $p(z_t|z_{t-1}, s_t, b_t)$ can only select a new option z_t if the previous option z_{t-1} has terminated.

3 METHOD

We aim to learn a reusable set of options that allow for faster training on new tasks from a given distribution. To differentiate ourselves from classical 'hard' options, which, once learned, do not change during new tasks, we call our novel approach *soft-options*. Each soft-option consists of an option *prior*, denoted by p_θ, which is shared across all tasks, and a task-specific option *posterior*, denoted by q_{ϕ_i} for task i. Unlike most previous work, e.g. (Frans et al., 2018), we learn both intra-option and termination policies. The priors of both the intra-option policy p^L_θ and the termination policy p^T_θ capture

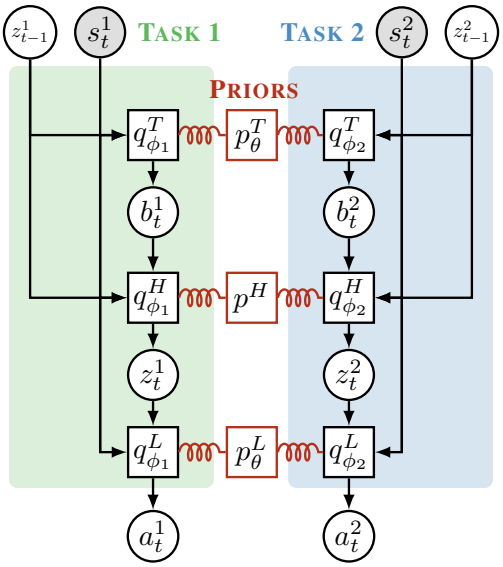


Figure 1: Two hierarchical posterior policies (left and right) with common priors (middle). For each task i, the policy conditions on the current state s^i_t and the last selected option z^i_{t-1}. It samples, in order, whether to terminate the last option (b^i_t), which option to execute next (z^i_t) and what primitive action (a^i_t) to execute in the environment.

how an option typically behaves and remain fixed once they are fully learned. At the beginning of training on a new task, they are used to initialize the task-specific posterior distributions $q^L_{\phi_i}$ and $q^T_{\phi_i}$. During training, the posterior is then regularized against the prior to prevent inadvertent unlearning. However, if maximizing the reward on certain tasks is not achievable with the prior policy, the posterior is free to deviate from it. We can thus speed up training using options, while remaining flexible enough to solve more tasks. Additionally, this soft option framework also allows for learning good *priors* in a multitask setting while avoiding complex training schedules and local optima (see Section 3.5). In this work, we also learn the higher-level posterior $q^H_{\phi_i}$ within the framework of PAI, but assume a fixed, uniform prior distribution p^H, i.e. we assume there is no shared higher-level structure between tasks. Figure 1 shows an overview over this architecture which we explain further below.

3.1 HIERARCHICAL POSTERIOR POLICIES

To express options in the PAI framework, we introduce two additional variables at each time step t: *option selections* z_t, representing the currently selected option, and decisions b_t to *terminate* them and allow the higher-level (master) policy to choose a new option. The agent's behavior depends on the currently selected option z_t, by drawing actions a_t from the *intra-option posterior policy* $q^L_{\phi_i}(a_t|s_t, z_t)$. The selection z_t itself is

drawn from a *master policy* $q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t) = (1 - b_t)\,\delta(z_t - z_{t-1}) + b_t\, q^H_{\phi_i}(z_t|s_t)$, which conditions on $b_t \in \{0, 1\}$, drawn by the *termination posterior policy* $q^T_{\phi_i}(b_t|s_t, z_{t-1})$. The master policy either continues with the previous z_{t-1} or draws a new option, where we set $b_1 = 1$ at the beginning of each episode. We slightly abuse notation by referring by $\delta(z_t - z_{t-1})$ to the Kronecker delta $\delta_{z_t, z_{t-1}}$ for discrete and the Dirac delta distribution for continuous z_t. The joint posterior policy is

$$q_{\phi_i}(a_t, z_t, b_t|s_t, z_{t-1}) = \\ q^T_{\phi_i}(b_t|s_t, z_{t-1})\, q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)\, q^L_{\phi_i}(a_t|s_t, z_t)\,. \tag{4}$$

While z_t can be a continuous variable, we consider only $z_t \in \{1 \dots m\}$, where m is the number of available options. The induced distribution $q_{\phi_i}(\tau)$ over trajectories of task i, $\tau = (s_1, b_1, z_1, a_1, s_2, \dots, s_T, b_T, z_T, a_T, s_{T+1})$, is then

$$q_{\phi_i}(\tau) = \rho_i(s_1) \prod_{t=1}^{T} q_{\phi_i}(a_t, z_t, b_t|s_t, z_{t-1}) P_i(s_{t+1}|s_t, a_t). \tag{5}$$

3.2 HIERARCHICAL PRIOR POLICY

Our framework transfers knowledge between tasks by a shared prior $p_\theta(a_t, z_t, b_t|s_t, z_{t-1})$ over all joint policies (4):

$$p_\theta(a_t, z_t, b_t|s_t, z_{t-1}) = \\ p^T_\theta(b_t|s_t, z_{t-1})\, p^H(z_t|z_{t-1}, b_t)\, p^L_\theta(a_t|s_t, z_t). \tag{6}$$

By choosing p^T_θ, p^H, and p^L_θ correctly, we can learn useful temporally extended options. The parameterized priors $p^T_\theta(b_t|s_t, z_{t-1})$ and $p^L_\theta(a_t|s_t, z_t)$ are structurally equivalent to the posterior policies $q^T_{\phi_i}$ and $q^L_{\phi_i}$ so that they can be used as initialization for the latter on new tasks. Optimizing the regularized return (see next section) w.r.t. θ distills the common behavior into the prior policy and softly enforces similarity across posterior distributions of each option amongst all tasks i.

The prior $p^H(z_t|z_{t-1}, b_t) = (1 - b_t)\,\delta(z_t - z_{t-1}) + b_t \frac{1}{m}$ selects the previous option z_{t-1} if $b_t = 0$, and otherwise draws options uniformly to ensure exploration. Because the posterior master policy is different on each task, there is no need to distill common behavior into a joint prior.

3.3 OBJECTIVE

We extend the multitask objective in (3) by substituting $p_\theta(\tau, \mathcal{O}_{1:T})$ and $p_{\phi_i}(\tau)$ with those induced by our hierarchical posterior policy in (4) and the corresponding prior. The resulting objective has the same form but with a new regularized reward that is maximized:

$$R^{\text{reg}}_{i,t} = r_i(s_t, a_t) - \underbrace{\beta \ln \frac{q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)}{p^H(z_t|z_{t-1}, b_t)}}_{①} \\ - \underbrace{\beta \ln \frac{q^L_{\phi_i}(a_t|s_t, z_t)}{p^L_\theta(a_t|s_t, z_t)}}_{②} - \underbrace{\beta \ln \frac{q^T_{\phi_i}(b_t|s_t, z_{t-1})}{p^T_\theta(b_t|s_t, z_{t-1})}}_{③}. \tag{7}$$

As we maximize $\mathbb{E}_q[R^{\text{reg}}_{i,t}]$, this corresponds to maximizing the expectation over

$$r_i(s_t, a_t) - \beta \big[\mathbb{D}_{\text{KL}}(q^H_{\phi_i} \| p^H) + \mathbb{D}_{\text{KL}}(q^L_{\phi_i} \| p^L_\theta) + \mathbb{D}_{\text{KL}}(q^T_{\phi_i} \| p^T_\theta)\big], \tag{8}$$

along the on-policy trajectories drawn from $q_{\phi_i}(\tau)$. In the following, we will discuss the effects of all three regularization terms on the optimization.

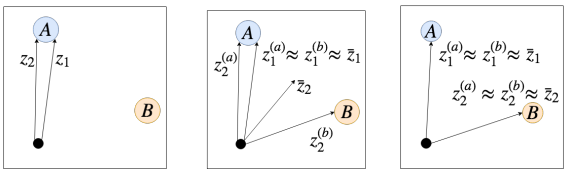
Term ① of the regularization encourages exploration in the space of options since we chose a uniform prior for p^H when the previous option was terminated. It can *also* be seen as a form of deliberation cost (Harb et al., 2017) as it is only nonzero whenever we terminate an option and the master policy needs to select another to execute: if the option is not terminated, we have $z_t = z_{t-1}$ with probability 1 for both prior and posterior by construction and $\mathbb{D}_{\text{KL}}(q^H_{\phi_i} \| p^H) = 0$.

Because (7) is optimized across *all* tasks i, term ② updates the prior towards the 'average' posterior. It also regularizes each posterior towards this prior. This enforces similarity between option posteriors across tasks. Importantly, it also encourages the *master* policy to pick the most *specialized* option that still maximizes the return, i.e the option for which the posteriors $q^L_{\phi_i}$ are most similar across tasks as this will minimize term ②. Consequently, if multiple options have learned the desired behavior, the master policy will only pick the most specialized option consistently. As discussed in Section 3.5, this allows us to escape the local optima that hard options face in multitask learning, while still having fully specialized options after training.

Lastly, we can use ③ to also encourage temporal abstraction of options. To do so, *during option learning*, we fix the termination prior p^T to a Bernoulli distribution $p^T(b) = (1 - \alpha)^b \alpha^{1-b}$. Choosing a large α encourages prolonged execution of one option, but allows switching whenever necessary. This is similar to deliberation costs (Harb et al., 2017) but with a more flexible cost model.

We can still distill a termination prior p^T_θ which can be used on future tasks. Instead of learning p^T_θ by minimizing the KL against the posterior termination policies, we can get more decisive terminations by minimizing

$$\min_\theta \sum_{i=1}^{n} \mathbb{E}_{\tau \sim q^i} \big[\mathbb{D}_{\text{KL}}\big(\hat{q}_{\phi_i}(\cdot|s_t, z_{t-1}) \| p^T_\theta(\cdot|s_t, z_{t-1})\big)\big], \tag{9}$$



(a) Hard options (b) Soft options (c) After training

Figure 2: Hierarchical learning of two concurrent tasks (a and b) using two options (z_1 and z_2) to reach two relevant targets (A and B). a) Local optimum when simply sharing options across tasks. b) Escaping the local optimum by using prior ($\bar{z}_i$) and posterior ($z_i^{(j)}$) policies. c) Learned options after training. Details are given in the text in Section 3.5.

and $\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) = \sum_{z_t \neq z_{t-1}} q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t=1)$ i.e., the learned termination prior distills the probability that the tasks' master policies would change the active option if they had the opportunity. Details on how we optimized the MSOL objective are given in Appendix B.

3.4 MSOL VS. CLASSICAL OPTIONS

Assume we are faced with a new task and are given some prior knowledge in the form of a set of skills that we can use. Using the skills' policies and termination probabilities as prior policies p^T and p^L in the soft option framework, we can interpret β as a temperature parameter determining how closely we are required to follow them. For $\beta \to \infty$ we recover the classical "hard" option case and our posterior option policies are restricted to the prior.[1] For $\beta = 0$ the priors only initialize the otherwise unconstrained policy, quickly unlearning behavior that may be useful down the line. Only for $0 < \beta < \infty$ MSOL can keep prior information to guide long-term exploration but can also explore policies "close" to them.

3.5 LOCAL OPTIMA OPTION LEARNING

In this section our aim is to provide an intuitive explanation of why learning hard options in a multitask setting can lead to local optima and how soft options can overcome this. In this local optimum, multiple options have learned the same behavior and are unable to change it, even if doing so would ultimately lead to a higher reward. We use the Moving Bandits experiment schematically depicted in Figure 2 as an example. The agent (black dot) observes two target locations A and B but does not know which one is the correct one that has to be reached in order to generate a reward. The state- and action-spaces are continuous, requiring multiple actions to reach either A or B from the starting position. Consequently, having access to two options, one for each location, can accelerate learning. Experimental results comparing MSOL against a recently proposed 'hard option' method (Meta Learning of Shared Hierarchies (MLSH), (Frans et al., 2018)) are discussed in Section 5.1.

Let us denote the options we are learning as z_1 and z_2 and further assume that due to random initialization or late discovery of target B, both skills currently reach A. In this situation, the master policies on tasks in which the correct goal is A are indifferent between using z_1 and z_2 and will consequently use *both* with equal probability.

In the case of hard options, changing one skill, e.g. z_2, towards B in order to solve tasks in which B is the correct target, decreases the performance on all tasks that currently use z_2 to reach target A, because for hard options the skills are shared exactly across tasks. Averaged across all tasks, this would at first *decrease* the overall average return, preventing any option from changing away from A, leaving B unreachable and training stuck in a local optimum.

To "free up" z_2 and learn a new skill reaching B, all master policies need to refrain from using z_2 to reach A and instead use the equally useful skill z_1 exclusively. Importantly, using soft options makes this possible. In Figures 2(b) and 2(c) we depict this schematically. The key difference is that in MSOL we have separate *task-specific posteriors* $z_i^{(a)}$ and $z_i^{(b)}$ for tasks a and b and soft options $i \in \{1, 2\}$ (for simplicity, we assume that the correct target is A for task a and B for task b). This allows us, in a first step, to solve *all* tasks (Figure 2(b)): despite master policies on tasks a still using posterior $z_2^{(a)}$ to reach A, the other posterior $z_2^{(b)}$ can learn to reach B. However, this now makes option z_2 less specialized across tasks, i.e. the prior $\bar{z}_2$ does not agree with either posterior $z_2^{(a)/(b)}$. Consequently, for tasks a, the master policies will now strictly prefer option z_1 to reach A, allowing option z_2 to specialize on only reaching B, leading to the situation shown in Figure 2(c) in which both options specialize to reach different targets.

4 RELATED WORK

Most hierarchical approaches rely on proxy rewards to train the lower level components and their terminations. Some of them aim to reach pre-specified subgoals (Sutton et al., 1999), which are often found by analyzing the structure of the MDP (McGovern and Barto, 2001), previously learned policies (Tessler et al., 2017) or predictability (Harutyunyan et al., 2019). Those methods typically require knowledge, or a sufficient approximation, of the transition model, both of which are often infeasible.

Recently, several authors have proposed unsupervised

[1] However, in this limiting case optimization using the regularized reward is not possible.

training objectives for learning diverse skills based on their distinctiveness (Gregor et al., 2016). However, those approaches don't learn termination functions and cannot guarantee that the required behavior on the downstream task is included in the set of learned skills. Hausman et al. (2018) also incorporate reward information, but do not learn termination policies and are therefore restricted to learning multiple solutions to the provided task instead of learning a *decomposition* of the task solutions which can be re-composed to solve new tasks.

A third usage of proxy rewards is by training lower level policies to move towards goals defined by the higher levels. When those goals are set in the original state space (Nachum et al., 2018), this approach has difficulty scaling to high dimensional state spaces like images. Setting the goals in a learned embedding space (Vezhnevets et al., 2017) can be difficult to train, though. In both cases, the temporal extension of the learned skills are set manually. On the other hand, Goyal et al. (2019) also learn a hierarchical agent, but not to transfer skills, but to find decisions states based on how much information is encoded in the latent layer.

HiREPS Daniel et al. (2012) also take an inference motivated approach to learning options. In particular Daniel et al. (2016) propose a similarly structured hierarchical policy, albeit in a single task setting. However, they do not utilize learned prior *and* posterior distributions, but instead use expectation maximization to iteratively infer a hierarchical policy to explain the current reward-weighted trajectory distribution.

Several previous works try to overcome the restrictive nature of options that can lead to sub-optimal solutions by allowing the higher-level actions to modulate the behavior of the lower-level policies Heess et al. (2016); Haarnoja et al. (2018). However, this significantly increases the required complexity of the higher-level policy and therefore the learning time.

The multitask- and transfer-learning setup used in this work is inspired by Thrun and Schwartz (1995) who suggests extracting options by using commonalities between solutions to multiple tasks. Prior multitask approaches often rely on additional human supervision like policy sketches (Andreas et al., 2017) or desirable sub-goals (Tessler et al., 2017) in order to learn skills which transfer well between tasks. In contrast, our work aims at finding good termination states without such supervision. Tirumala et al. (2019) investigate the use of different priors for the higher-level policy while we are focussing on learning transferrable option priors. Closest to our work is MLSH (Frans et al., 2018) which, however, shares the lower-level policies across all tasks without distinguishing between prior and posterior and does not learn termination policies. As discussed, this leads to local minima and insufficient diversity in the learned options. Similarly to us, Fox et al. (2016) differentiate between prior and posterior policies on multiple tasks and utilize a KL-divergence between them for training. However, they do not consider termination probabilities and instead only choose one option per task.

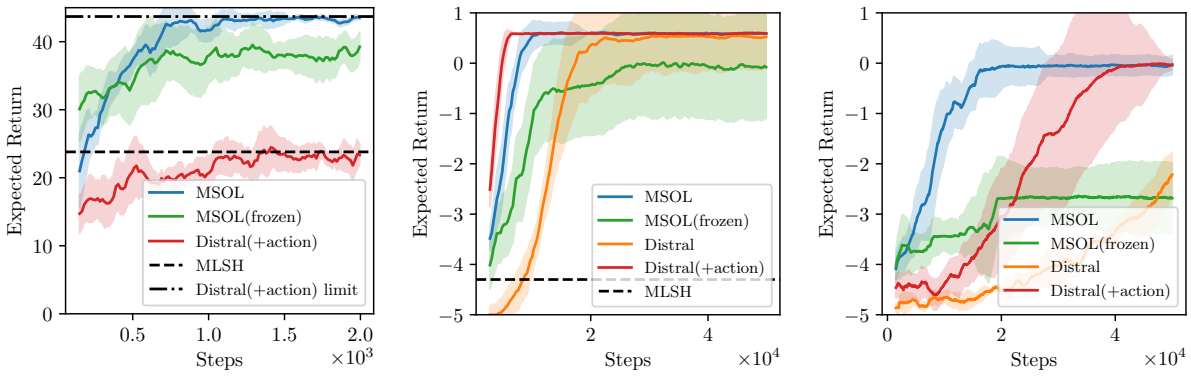
Our approach is closely related to DISTRAL (Teh et al., 2017) with which we share the multitask learning of prior and posterior policies. However, DISTRAL has no hierarchical structure and applies the same prior distribution over primitive actions, independent of the task. As a necessary hierarchical heuristic, the authors propose to also condition on the last primitive action taken. This works well when the last action is indicative of future behavior; however, in Section 5 we show several failure cases where a *learned* hierarchy is needed.

5 EXPERIMENTS

We conduct a series of experiments to show: i) MSOL trains successfully without complex training schedules like in MLSH (Frans et al., 2018), ii) MSOL can learn useful termination policies, iii) when learning hierarchies in a multitask setting, unlike other methods, MSOL successfully overcomes the local minimum of insufficient option diversity, as described in Section 3.5, iv) using soft options yields fast transfer learning while still reaching optimal performance, even on new, out-of-distribution tasks.

All architectural details and hyper-parameters can be found in the appendix. For all experiments, we first train the exploration priors and options on n tasks from the available task distribution $\mathcal{T}$ (training phase is plotted in Appendix D). Subsequently, we test how quickly we can learn new tasks from $\mathcal{T}$ (or another distribution $\mathcal{T}'$).

We compare the following algorithms: MSOL is our proposed method that utilizes soft options both during option learning and transfer. MSOL(frozen) uses the soft options framework during learning to find more diverse skills, but does not allow fine-tuning the posterior sub-policies after transfer. DISTRAL (Teh et al., 2017) is a strong non-hierarchical transfer learning algorithm that also utilizes prior and posterior distributions. DISTRAL(+action) utilizes the last action as option-heuristic, that is, as additional input to the policy and prior, which works well in some tasks but fails when the last action is not sufficiently informative. Conditioning on an *informative* last action allows the DISTRAL prior to learn temporally correlated exploration strategies. MLSH (Frans et al., 2018) is a multitask option learning algorithm like MSOL, but utilizes 'hard' options for both learning and transfer, i.e., sub-policies that are shared exactly across



(a) Moving Bandits (b) Taxi (c) Directional Taxi

Figure 3: Performance of applying the learned options and exploration priors to new tasks. Each line is the median over 5 random seeds (2 for MLSH) and shaded areas indicated standard deviations. Performance during the training phase is shown in Figure 6. *Moving Bandits* (a) is a simple environment capturing the effects described in Section 3.5. The results show that MLSH, which uses hard options, struggles with local minima during the learning phase, whereas MSOL is able to learn a diverse set of options. *Taxi* (b) and *Directional Taxi* (c) additionally require good termination policies, which MLSH cannot learn as it uses a fixed option duration. See Figure 4 for a visualization of the options and terminations learned by MSOL. DISTRAL(+action) is a strong non-hierarchical baseline which uses the last action as option-heuristic, but suffers when that action is not very informative, for example in (c).

tasks. It relies on fixed option durations and requires a complex training schedule between master and intra-option policies to stabilize training. We use the author's MLSH implementation. We also compare against Option Critic (OC) (Bacon et al., 2017), which takes the task-id as additional input in order to apply it to multiple tasks.

Note that, during test time, MLSH and MSOL(frozen) can be fairly compared as each uses one fixed policy per skill. On the other hand, DISTRAL, DISTRAL(+action) and MSOL use adaptive posterior policies for each task and are consequently more expressive.

5.1 MOVING BANDITS

We start with the 2D Moving Bandits environment proposed and implemented by Frans et al. (2018), which is similar to the example in Section 3.5. There are two randomly sampled, distinguishable, marked positions in the environment. In each episode, the agent receives a reward of 1 for each time step it is sufficiently close to the correct one of both positions, and 0 otherwise. Which location is rewarded is not signaled in the observation. The agent can take actions that move it in one of the four cardinal directions. Each episode lasts 50 steps.

We compare against MLSH and DISTRAL to highlight challenges that arise in multitask training. We allow MLSH and MSOL to learn two options. During transfer, optimal performance can only be achieved with diverse options that have successfully learned to reach *different* marked locations. In Figure 3(a) we can see that MSOL is able to do so but the hard options learned by MLSH both learned to reach the *same* goal location, resulting in only approximately half the optimal return during transfer. This is exactly the situation outlined in Section 3.5 in which learning hard options can lead to local optima.

DISTRAL, even with the last action provided as additional input, is not able to quickly utilize the prior knowledge. The last action only conveys meaningful information when taking the goal locations into account: DISTRAL agents need to *infer* the intention based on the last action and the relative goal positions. While this is possible, in practice the agent was not able to do so, even with a much larger network. Much longer training ultimately allows DISTRAL to perform as well as MSOL, denoted by "DISTRAL(+action) limit". This is not surprising since its posterior is flexible and will therefore eventually be able to learn any task. However, it is not able to learn transferrable prior knowledge which allows *fast* training on the new task. Lastly, MSOL(frozen) also outperforms DISTRAL(+action) and MLSH, but performs worse that MSOL. This highlights the utility of making options soft, i.e. adaptable, during transfer to new tasks. It also shows that the advantage of MSOL over the other methods lies not only in its flexibility during transfer, but also during the original learning phase.

5.2 TAXI

Next, we use a slightly modified version of the original Taxi domain (Dietterich, 1998) to show learning of termination functions as well as transfer- and generalization capabilities. To solve the task, the agent must pick up a passenger on one of four possible locations by moving

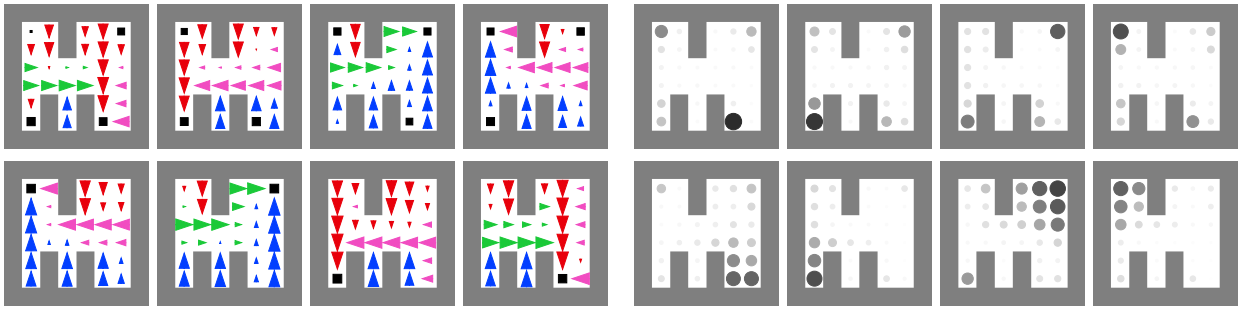


Figure 4: Options learned with MSOL on the taxi domain, before (top) and after pickup (bottom). The light gray area indicates walls. The left plots show the intra-option policies: arrows and colors indicated direction of most likely action, the size indicates its probability. A square indicates the pickup/dropoff action. The right plots show the termination policies: intensity and size of the circles indicate termination probability.

to their location and executing a special 'pickup/drop-off' action. Then, the passenger must be dropped off at one of the other three locations, again using the same action executed at the corresponding location. The domain has a discrete state space with 30 locations arranged on a grid and a flag indicating whether the passenger was already picked up. The observation is a one-hot encoding of the discrete state, excluding passenger- and goal location. This introduces an information-asymmetry between the task-specific master policy, and the shared options, allowing them to generalize well (Galashov et al., 2019). Walls (see Figure 4) limit the movement of the agent and invalid actions.

We investigate two versions of Taxi. In the original, just called *Taxi*, the action space consists of one no-op, one 'pickup/drop-off' action and four actions to move in all cardinal directions. In *Directional Taxi*, we extend this setup: the agent faces in one of the cardinal directions and the available movements are to move forward or rotate either clockwise or counter-clockwise. In both environments the set of tasks $\mathcal{T}$ are the 12 different combinations of pickup/drop-off locations. Episodes last at most 50 steps and there is a reward of 2 for delivering the passenger to its goal and a penalty of -0.1 for each time step. During training, the agent is initialized to any valid state. During testing, the agent is always initialized without the passenger on board.

We allow four learnable options in MLSH and MSOL. This necessitates the options to be diverse, i.e., one option to reach each of the four pickup/drop-off locations. Importantly, it also requires the options to learn to terminate when a passenger is picked up. As one can see in Figure 3(b), MLSH struggles both with option-diversity and due to its fixed option duration: because the starting position is random, the duration until the option needs to terminate is different between episodes and cannot be captured by one hyperparameter. Furthermore, even without correct terminations, one could still learn to solve (at least) four out of the twelve tasks, leading to an average reward of approximately -3.2[2]. However, MLSH is not able to learn diverse enough policies, resulting in worse performance.

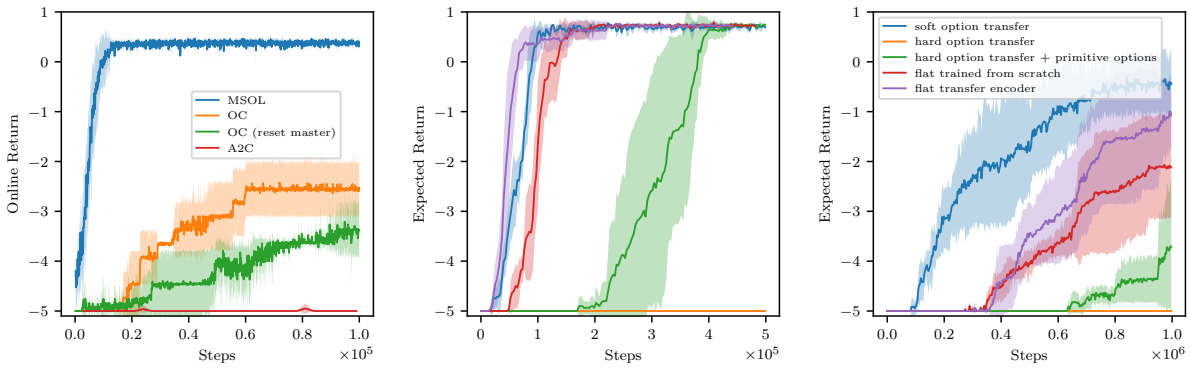
DISTRAL(+action) performs well in the original *Taxi* environment, as seen in Figure 3(b). This is expected since here the last action, moving in a compass direction, is a good indicator for the agent's intention, effectively acting as an optimal "option" and inducing temporally extended exploration. However, in the directional case shown in Figure 3(c), actions rarely indicate intentions, which makes it much harder for DISTRAL(+action) to use prior knowledge. By contrast, MSOL performs well in both taxi environments. In the directional case, learned MSOL options capture temporally correlated behavior much better than the last action in DISTRAL.

Figure 4 demonstrates that the options learned by MSOL learn movement and termination policies that make intuitive sense. Note that the same soft option represents different behavior depending on whether it already picked up the passenger, as this behavior does not need to terminate the current option on three of the 12 tasks.

5.3 OUT-OF-DISTRIBUTION TASKS

In this section, we show how learning soft options can help with transfer to unseen tasks. In Figure 5(a) we show learning on four tasks from $\mathcal{T}$ using options that were trained on the remaining eight, comparing against Advantage Actor-critic (A2C) (Mnih et al., 2016) and Option Critic (OC) (Bacon et al., 2017). Note that in OC, there is no information-asymmetry: the same networks are shared across all tasks and provided with a task-id as additional input, including to the option-policies. This prevents OC from generalizing well to unseen tasks. On the other hand, withholding the task-information would

[2]The optimal policy for a task achieves approximate a return of 0.5 on average whereas the worst possible return is -5.



(a) Taxi: Generalization (b) Taxi (small): Adaptation (c) Taxi (large): Adaptation

Figure 5: We compare MSOL against Option Critic (OC), hard options and flat policies trained from scratch or with a pre-trained encoder. For a fair comparison, the soft option prior is identical to the hard option in these experiments. *Left:* Since the options in OC are not task-agnostic, they fail to generalize to previously unseen tasks. *Middle and right:* Transfer performance of options to environments in which the pickup and dropoff locations where shifted, making the options misspecified. Only soft options provide utility over flat policies in this setting. The middle figure shows results on a small grid in which exploration is simple, whereas the right figure shows that transfer learning can accelerate exploration especially on larger tasks.

be similar to MLSH, which we already showed to struggle with local minima. The strong performance of MSOL shows that information-asymmetric options help to generalize to previously unseen tasks.

We also investigate the utility of flexible soft options under a shift of the task distribution: in Figures 5(b) and 5(c) we show learning performance on twelve *modified* tasks in which the pickup/dropoff locations where moved by one cell while the options were trained with the original locations. While the results in Figure 5(b) use a smaller grid, Figure 5(c) shows the results for a larger grid in which exploration is more difficult. As expected, hard options are not able to solve this task for either grid-size. Moreover, while combining hard options with primitive actions allows the tasks to be solved eventually, it performs worse than training a new, flat policy from scratch. The finding that access to misspecified, hard options can actually *hurt* exploration is consistent with previous literature (Jong et al., 2008). On the other hand, MSOL is able to quickly learn on this new task by adapting the previously learned options.

Note that on the small grid in which exploration is easy, our hierarchical method performs similar to a flat policy. On the larger grid exploration becomes more challenging and MSOL learns significantly faster, highlighting how transfer learning can improve exploration. More results can be found in Appendix D.2.

6 DISCUSSION

Multitask Soft Option Learning (MSOL) proposes reformulating options using the perspective of prior and posterior distributions. This offers several key advantages. First, during transfer, it allows us to distinguish between fixed, and therefore knowledge-preserving option *priors*, and flexible option *posteriors* that can adjust to the reward structure of the task at hand. This effects a similar speed-up in learning as the original options framework, while avoiding sub-optimal performance when the available options are not perfectly aligned to the task. Second, utilizing this 'soft' version of options in a multitask learning setup increases optimization stability and removes the need for complex training schedules. Furthermore, this framework naturally allows master policies to coordinate across tasks and avoid local minima of insufficient option diversity. It also allows for autonomously learning option-termination policies, a very challenging task which is often avoided by fixing option durations manually.

Lastly, using this formulation also allows inclusion of prior information in a principled manner without imposing too rigid a structure on the resulting hierarchy. We utilize this advantage to explicitly incorporate the bias that good options should be temporally extended. In future research, other types of information can be explored. As an example, one could investigate sets of tasks which would benefit from a learned master prior, like walking on different types of terrain.

7 ACKNOWLEDGEMENTS

MI is supported by the AIMS EPSRC CDT. NS, AG, NN were funded by ERC grants ERC-2012-AdG 321162-HELIOS and Seebibyte EP/M013774/1, and EPSRC/MURI grant EP/N019474/1. SW is supported by ERC under the Horizon 2020 research and innovation programme (grant agreement number 637713).

References

J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *AAAI*, 2017.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. ACM, 2009.

M. M. Botvinick, Y. Niv, and A. C. Barto. Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, (3), 2009.

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

C. Daniel, H. Van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 2016.

T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *ICML*, 1998.

R. Fox, M. Moshkovitz, and N. Tishby. Principled option learning in markov decision processes. *arXiv:1609.05524*, 2016.

K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. In *ICLR*, 2018.

A. Galashov, S. Jayakumar, L. Hasenclever, D. Tirumala, J. Schwarz, G. Desjardins, W. M. Czarnecki, Y. W. Teh, R. Pascanu, and N. Heess. Information asymmetry in KL-regularized RL. In *ICLR*, 2019.

A. Goyal, R. Islam, D. Strouse, Z. Ahmed, H. Larochelle, M. Botvinick, S. Levine, and Y. Bengio. Transfer and exploration via the information bottleneck. In *ICLR*, 2019.

K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *arXiv:1611.07507*, 2016.

T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine. Latent space policies for hierarchical reinforcement learning. *arXiv:1804.02808*, 2018.

J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup. When waiting is not an option: Learning options with a deliberation cost. *arXiv:1709.04571*, 2017.

A. Harutyunyan, W. Dabney, D. Borsa, N. Heess, R. Munos, and D. Precup. The termination critic. *arXiv:1902.09996*, 2019.

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

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

N. K. Jong, T. Hester, and P. Stone. The utility of temporal abstraction in reinforcement learning. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv:1805.00909*, 2018.

A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*, 2001.

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

O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *arXiv:1805.08296*, 2018.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

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

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

Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. In *NeurIPS*, 2017.

C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, 2017.

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *NeurIPS*, 1995.

D. Tirumala, H. Noh, A. Galashov, L. Hasenclever, A. Ahuja, G. Wayne, R. Pascanu, Y. W. Teh, and N. Heess. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv:1903.07438*, 2019.

E. Todorov. General duality between optimal control and estimation. IEEE, 2008.

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

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv:1611.05763*, 2016.

APPENDIX

A PSEUDO CODE

Algorithm 1: Pseudo-Code for MSOL

Input Number m of options to learn, n_i different training tasks env_i, *fixed* termination prior $p^T(b_t) = (1-\alpha)^{b_t}\alpha^{1-b_t}$ and *fixed* master prior $p^H(z_t|z_{t-1}, b_t) = (1-b_t)\,\delta(z_t - z_{t-1}) + b_t \frac{1}{m}$

Initialize once: *Learnable* termination prior $p^T_\theta(b_t|s_t, z_{t-1})$, intra-option prior $p^L_\theta(a_t|s_t, z_t)$

Initialize for each task i**:** Lernable termination posterior $q^T_{\phi_i}(b_t|s_t, z_{t-1})$, master policy $q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)$, intra-option policy $q^L_{\phi_i}(a_t|s_t, z_t)$

// Note that this leads to a total of m intra option priors for the m different values of $z \in \{1 \dots m\}$

// and a total of $m \times n_i$ intra option posteriors.

while not converged **do**

  // Collect data

  for each task i **do**

    if beginning of episode **then**

      $s_0 \leftarrow \mathrm{env.reset}()$

      $b_0 \leftarrow 1$ *// This allows q^H to sample a new option z_0.*

    else

      $b_t \sim q^T_{\phi_i}(b_t|s_t, z_{t-1})$

    $z_t \sim q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)$

    $a_t \sim q^L_{\phi_i}(a_t|s_t, z_t)$

    $s_{t+1}, r_t \sim \mathrm{env}_i(a_t)$

    // Compute regularized reward (eq. (7)); note that we use the fixed priors here

    $R^{\mathrm{reg}}_t \leftarrow r_i(s_t, a_t) - \beta \ln \frac{q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)}{p^H(z_t|z_{t-1}, b_t)} - \beta \ln \frac{q^L_{\phi_i}(a_t|s_t, z_t)}{p^L_\theta(a_t|s_t, z_t)} - \beta \ln \frac{q^T_{\phi_i}(b_t|s_t, z_{t-1})}{p^T(b_t)}$

    Add $s_t, s_{t+1}, r_t, R^{\mathrm{reg}}_t$ to $\mathcal{D}_i$

  // Update parameters ϕ_i

  for each task i **do**

    Update ϕ_i using A2C or PPO on $\mathcal{D}_i$ as desribed in Appendix B.1. Note that for PPO, R^{reg}_t needs to be re-computed and updated between gradient updates to ϕ_i as the regularization terms change.

  // Update parameters θ

  Update θ to minimize

  $\sum_i \mathbb{E}_{\mathcal{D}_i}\left[\mathbb{D}_{\mathrm{KL}}\Big(q^L_{\phi_i}(a_t|s_t, z_t) \parallel p^L_\theta(a_t|s_t, z_t)\Big) + \mathbb{D}_{\mathrm{KL}}\big(\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) \parallel p^T_\theta(b_t|s_t, z_{t-1})\big)\right]$

  Here, $\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) = \sum_{z_t \neq z_{t-1}} q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t = 1)$, i.e. instead of distilling the average of $q^T_{\phi_i}$ into p^T_θ, we distill whether the *master policy* $q^H_{\phi_i}$ would have changed the option z_t if it had the chance (i.e. if $b_t = 1$). Since $q^T_{\phi_i}$ is regularized to be similar to the *fixed* p^T, this approach allows us to learn a termination prior p^T_θ which is less influenced by our manually specified prior p^T, and more by what is needed for the task.

B MSOL TRAINING DETAILS

B.1 OPTIMIZATION

Even though R_i^{reg} depends on ϕ_i, its gradient w.r.t. ϕ_i vanishes.[3] Consequently, we can treat the regularized reward as a classical RL reward and use any RL algorithm to find the optimal hierarchical policy parameters ϕ_i. In the following, we explain how to adapt A2C (Mnih et al., 2016) to soft options. The extension to PPO (Schulman et al., 2017) is straightforward.[4]

The joint posterior policy in (4) depends on the current state s_t and the previously selected option z_{t-1}. The expected sum of regularized future rewards of task i, the value function V_i, must therefore also condition on this pair:

$$V_i(s_t, z_{t-1}) \;:=\; \mathbb{E}_{\tau\sim q}\Big[\sum_{t'=t}^{T}\gamma^{t'-t}R_{i,t'}^{\text{reg}}\,\Big|\,s_t, z_{t-1}\Big]\,. \tag{10}$$

As $V_i(s_t, z_{t-1})$ cannot be directly observed, we approximate it with a parametrized model $V_{\phi_i}(s_t, z_{t-1})$. The k-step advantage estimation at time t of trajectory τ is given by

$$A_{\phi_i}(\tau_{t:(t+k)}) \quad := \quad \sum_{j=0}^{k-1}\gamma^j R_{t+j}^{\text{reg}} + \gamma^k V_{\phi_i}^{-}(s_{t+k}, z_{t+k-1}) - V_{\phi_i}(s_t, z_{t-1})\,, \tag{11}$$

where the superscript '$-$' indicates treating the term as a constant. The approximate value function V_{ϕ_i} can be optimized towards its bootstrapped k-step target by minimizing $\mathcal{L}_V(\phi_i, \tau_{1:T}) := \sum_{t=1}^{T}(A_{\phi_i}(\tau_{t:(t+k)}))^2$. As per A2C, $k \in [1 \dots n_s]$ depending on the state (Mnih et al., 2016). The corresponding policy gradient loss is

$$\mathcal{L}_A(\phi_i, \tau_{1:T}) := \sum_{t=1}^{T} A_{\phi_i}^{-}(\tau_{t:(t+k)}) \ln q_{\phi_i}(a_t, z_t, b_t | s_t, z_{t-1})\,.$$

The gradient w.r.t. the prior parameters θ is[5]

$$\nabla_\theta \mathcal{L}_P(\theta, \tau_{1:T}, \tilde{b}_{1:T}) \quad := \quad -\sum_{t=1}^{T}\Big(\nabla_\theta \ln p_\theta^L(a_t|s_t, z_t) + \nabla_\theta \ln p_\theta^T(\tilde{b}_t|s_t, z_{t-1})\Big)\,, \tag{12}$$

where $\tilde{b}_t = \delta_{z_{t-1}}(z'_t)$ and $z'_t \sim q^H(z'_t|s_t, z_{t-1}, b_t = 1)$. To encourage exploration in all policies of the hierarchy, we also include an entropy maximization loss:

$$\mathcal{L}_H(\phi_i, \tau_{1:T}) \quad := \quad \sum_{t=1}^{T}\Big(\ln q_{\phi_i}^H(z_t|s_t, z_{t-1}, b_t) + \ln q_{\phi_i}^L(a_t|s_t, z_t) + \ln q_{\phi_i}^T(b_t|s_t, z_{t-1})\Big)\,. \tag{13}$$

Note that term ① in (7) already encourages maximizing $\mathcal{L}_H(\phi_i, \tau)$ for the master policy, since we chose a uniform prior $p^H(z_t|b_t = 1)$. As both terms serve the same purpose, we are free to drop either one of them. In our experiments, we chose to drop the term for q^H in R_t^{reg}, which proved slightly more stable to optimize that the alternative.

We can optimize all parameters jointly with a combined loss over all tasks i, based on sampled trajectories $\tau^i := \tau_{1:T}^i \sim q_{\phi_i}$ and corresponding sampled values of $\tilde{b}^i := \tilde{b}_{1:T}^i$:

$$\mathcal{L}(\{\phi_i\}, \theta, \{\tau^i\}, \{\tilde{b}^i\}) = \sum_{i=1}^{n}\Big(\mathcal{L}_A(\phi_i, \tau^i) + \lambda_V\mathcal{L}_V(\phi_i, \tau^i) + \lambda_P\mathcal{L}_P(\theta, \tau^i, \tilde{b}^i) + \lambda_H\mathcal{L}_H(\phi_i, \tau^i)\Big)\,.$$

B.2 TRAINING SCHEDULE

For faster training, it is important to prevent the master policies q^H from converging too quickly to allow sufficient updating of all options. On the other hand, a lower exploration rate leads to more clearly defined options. We consequently anneal the exploration bonus λ_H with a linear schedule during training.

Similarly, a high value of β leads to better options but can prevent finding the extrinsic reward $r_i(s_t, a_t)$ early on in training. Consequently, we increase β over the course of training, also using a linear schedule.

[3] $\int p(x)\,\nabla \ln p(x)\,dx = \int \nabla p(x)\,dx = \nabla \int p(x)\,dx = 0$.

[4] However, for PAI frameworks like ours, unlike in the original PPO implementation, the advantage function must be updated after each epoch.

[5] Here we ignore β as it is folded into λ_P later.

C ARCHITECTURE

All policies and value functions share the same encoder network with two fully connected hidden layers of size 64 for the Moving Bandits environment and three hidden layers of sizes 512, 256, and 512 for the Taxi environments. Distral was tested with both model sizes on the Moving Bandits task to make sure that limited capacity is not the problem. Both models resulted in similar performance, the results shown in the paper are for the larger model. Master-policies, as well as all prior- and posterior policies and value functions consist of only one layer which takes the latent embedding produced by the encoder as input. Furthermore, the encoder is shared across tasks, allowing for much faster training since observations can be batched together.

Options are specified as an additional one-hot encoded input to the corresponding network that is passed through a single 128 dimensional fully connected layer and concatenated to the state embedding before the last hidden layer. We implement the single-column architecture of Distral as a hierarchical policy with just one option and with a modified loss function that does not include terms for the master and termination policies. Our implementation builds on the A2C/PPO implementation by, and we use the implementation for MLSH that is provided by the authors (`https://github.com/openai/mlsh`).

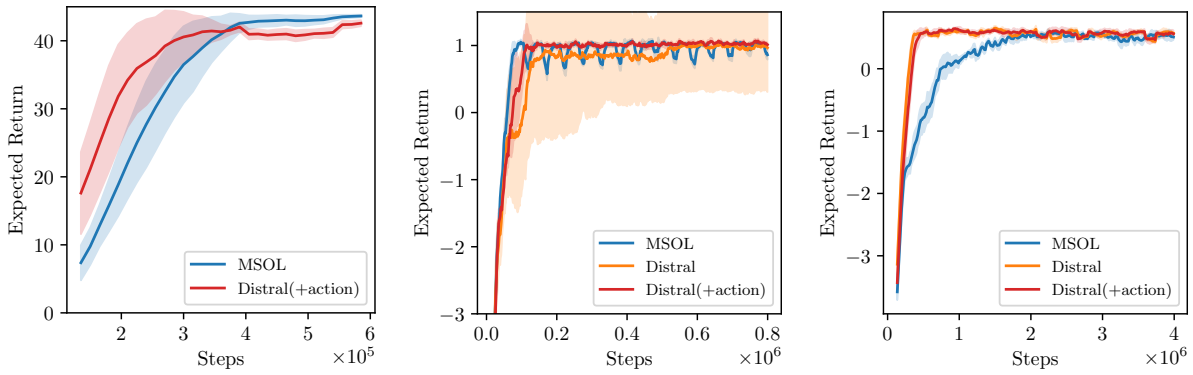
D HYPER-PARAMETERS AND ADDITIONAL ENVIRONMENT DETAILS

We use $2\lambda_V = \lambda_A = \lambda_P = 1$ in all experiments. Furthermore, we train on all tasks from the task distribution, regularly resetting individual tasks by resetting the corresponding master and re-initializing the posterior policies. Optimizing β for MSOL and Distral was done over $\{0.01, 0.02, 0.04, 0.1, 0.2, 0.4\}$. We use $\gamma = 0.95$ for Moving Bandits and Taxi.

D.1 MOVING BANDITS

For MLSH, we use the original hyper-parameters (Frans et al., 2018). The duration of each option is fixed to 10. The required warm-up duration is set to 9 and the training duration set to 1. We also use 30 parallel environments split between 10 tasks. This and the training duration are the main differences to the original paper. Originally, MLSH was trained on 120 parallel environments which we were unable to do due to hardware constraints. Training is done over 6 million frames per task.

For MSOL and Distral we use the same number of 10 tasks and 30 processes. The duration of options are learned and we do not require a warm-up period. We set the learning rate to 0.01 and $\beta = 0.2$, $\alpha = 0.95$, $\lambda_H = 0.05$. Training is done over 0.6 million frames per task. For Distral we use $\beta = 0.04$, $\lambda_H = 0.05$ and also 0.6 million frames per task.



(a) Moving Bandits (b) Taxi (c) Directional Taxi

Figure 6: Performance during training phase. Note that MSOL and MSOL(frozen) share the same training as they only differ during testing. Further, note that the highest achievable performance for Taxi and Directional Taxi is higher during training as they can be initialized closer to the final goal (i.e. with the passenger on board).

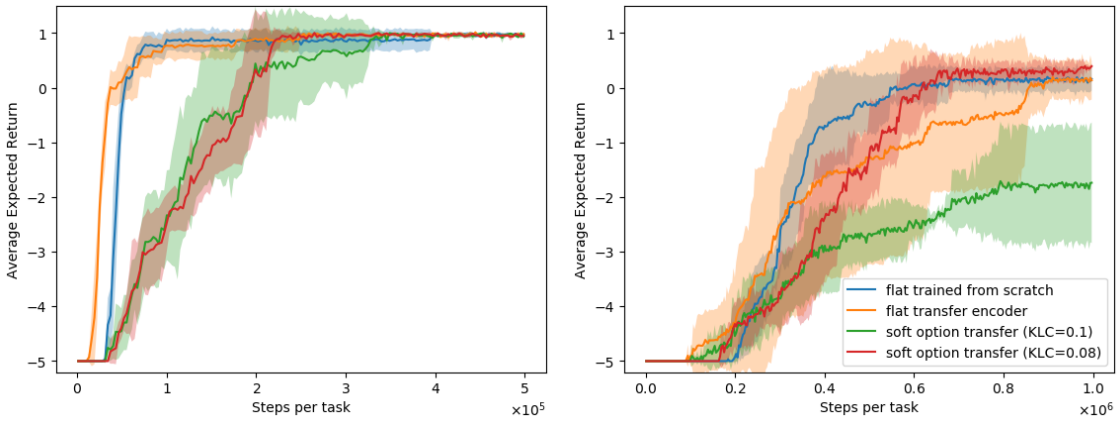


Figure 7: Results on a 'further modified' taxi environment in which the goal locations at test time were shifted compared to training, making the learned options misspecified, similar to Figures 5(b) and 5(c). Here, the goal locations were shifted further, making the options more misspecified. *Left:* Results on a 'small' 8x8 grid. *Right:* Results on a 'large' 10x10 grid.

D.2 TAXI

For MSOL we anneal β from 0.02 to 0.1 and λ_H from 0.1 to 0.05. For Distral we use $\beta = 0.04$. We use 3 processes per task to collect experience for a batch size of 15 per task. Training is done over 1.4 million frames per task for *Taxi* and 4 million frames per task for *Directional Taxi*. MLSH was trained on 0.6 million frames for *Taxi* as due to it's long runtime of several days, using more frames was infeasible. Training was already converged.

Tasks further out of distribution In Figure 7 we provided additional results for Section 5.3. We show the performance on *further modified* environments, for which the goal locations were moved by a second block from the original location for which the options were trained. We only compare soft options with flat policies trained from scratch, as we already showed in Section 5.3 that hard options are unable to cope well with goal modifications.

As expected, the flat policy trained from scratch performs similarly as before, as the moved goal location does not impact it much. On the other hand, using a pre-trained encoder performs slightly worse. On the smaller task (left figure) the options are too misspecified to be competitive, despite being soft. For the larger grid (right figure) and for a sufficienlty small value of β ('KLC'), the options, despite misspecification, are still competitive.

Consequently, while there is no hard limitation of our approach for appropriately chosen β, if the target task is too different from the source task, it will be faster to learn a new policy from scratch. Which algorithm trains faster depends mainly on the difficulty of exploration in the target task. Hard exploration makes options more useful compared to a new, flat policy, even if the options are misspecified. However, the more misspecified the options are, the smaller the advantage. If target and source task are too different, very little positive transfer can be expected, and learning a new (flat) policy becomes more efficient.