Journal of Machine Learning Research 17 (2016) 1-50 Submitted 5/15; Revised 11/15; Published 6/16

Hierarchical Relative Entropy Policy Search

Christian Daniel[1] DANIEL@IAS.TU-DARMSTADT.DE
Gerhard Neumann[1] NEUMANN@IAS.TU-DARMSTADT.DE
Oliver Kroemer[1] KROEMER@IAS.TU-DARMSTADT.DE
Jan Peters[1,2] PETERS@IAS.TU-DARMSTADT.DE

[1] *Technische Universität Darmstadt*
Fachbereich Informatik , Fachgruppe Intelligente Autonome Systeme
Hochschulstraße 10
64289 Darmstadt
Germany

[2] *Max-Planck-Institut für Intelligente Systeme*
Spemannstraße 38
72076 Tübingen
Germany

Editor: Laurent Orseau

Abstract

Many reinforcement learning (RL) tasks, especially in robotics, consist of multiple sub-tasks that are strongly structured. Such task structures can be exploited by incorporating hierarchical policies that consist of gating networks and sub-policies. However, this concept has only been partially explored for real world settings and complete methods, derived from first principles, are needed. Real world settings are challenging due to large and continuous state-action spaces that are prohibitive for exhaustive sampling methods. We define the problem of learning sub-policies in continuous state action spaces as finding a hierarchical policy that is composed of a high-level gating policy to select the low-level sub-policies for execution by the agent. In order to efficiently share experience with all sub-policies, also called inter-policy learning, we treat these sub-policies as latent variables which allows for distribution of the update information between the sub-policies. We present three different variants of our algorithm, designed to be suitable for a wide variety of real world robot learning tasks and evaluate our algorithms in two real robot learning scenarios as well as several simulations and comparisons.

Keywords: Reinforcement Learning, Policy Search, Hierarchical Learning, Robot Learning, Motor Skill Learning, Robust Learning, Structured Learning, Temporal Correlation, HiREPS, REPS

1. Introduction

Employing robots in unpredictable environments, such as in hospitals, disaster sites or households, requires robotic agents to autonomously learn new tasks and adapt to new environments. Such robots will need to acquire new skills through trial and error, also known as reinforcement learning (Sutton and Barto, 1998), as well as being able to generalize their skills to solve a large variety of tasks. However, successful implementation of reinforcement learning (RL) methods on real robot tasks is challenging for multiple reasons. Most importantly, real robots have high dimensional and continuous state-action spaces which is difficult to deal with for many RL methods. Furthermore, evaluations on real robots are resource intensive and, hence, the methods need to be sample efficient. Methods should also not fully explore the state-action space due to the risk of damaging the robot or its environment. Finally, real robot RL does not allow resetting of the state to arbitrary initial conditions as is required for many RL methods.

While presenting additional challenges, robot tasks also offer some advantages over traditional RL settings as many real world motor tasks are heavily structured. Exploiting the environment's structure can drastically simplify the learning problem. First, the solutions of many tasks lie within a tube of the solution space (Peters and Schaal, 2006), such that learning can be given a head start by demonstrating a sub-optimal solution. Local RL methods can, subsequently, be used to improve upon this demonstration. Second, the motor commands for many motor tasks exhibit strong temporal correlations. Such correlations allow for a hierarchical decomposition of the task into a sequence of elemental movements, often also referred to movement primitives (Schaal et al., 2003; Ijspeert and Schaal, 2003), options (Sutton et al., 1999b) or motor templates (Neumann and Peters, 2009). For example, a tennis game can be decomposed into a sequence of single strokes, e.g., a backhand strokes, forehand strokes, lobs, volleys and a serve. Finally, many motor tasks can be solved in multiple, often incompatible, ways. Identifying and representing such solutions as separate sub-policies to be used by the agent increases the versatility as well as the robustness of the learned policy. Additionally, it simplifies the use of local learning methods. In this paper, we extend our work on a robot learning framework (Daniel et al., 2012a,b), (Daniel et al., 2013) that can take advantage of such structured environments by identifying multiple sub-policies for a given task and learning to adapt, sequence and combine these sub-policies.

We base our learning algorithm on the Relative Entropy Policy Search (REPS) algorithm (Peters et al., 2010). In REPS, the exploitation-exploration trade-off is balanced by bounding the loss of information between policy updates. Such a bound results in a smoother and more stable learning process that avoids wild exploration of the state action space, as required by the robotics domain. We extend the REPS algorithm such that we can use sub-policies as building blocks of a hierarchical policy to exploit the temporal correlations inherent to many tasks. We formulate the problem of inferring such a hierarchical policy as a latent variable estimation problem, where the index of the sub-policy that has generated a given action is modeled as a latent variable. The policy update is now performed in two steps. In the Expectation step, we compute the responsibilities of the latent variables, i.e., the probabilities that the individual sub-policies have generated the observed state action pairs. The sub-policies are kept fixed in the E-step. Subsequently, the responsibilities are used to update the hierarchical policy containing the sub-policies with the REPS algorithm in the Maximization step. We show that such an Expectation-Maximization (EM) algorithm (Dempster et al., 1977) improves a lower bound of the original REPS optimization problem and that the lower bound is tight after each expectation step.

Our algorithm also allows for using prior knowledge to constrain the structure of the estimated hierarchical policy. For example, we often want the sub-policies to represent individual, distinct solutions. In many cases, these solutions may be incompatible, i.e., the solution space may not be convex. Therefore, the set of sub-policies should be separable in the solution space. Adding a constraint that avoids such an overlap ensures that the policy search algorithm does not average over multiple modes of the solution space, which may result in a poor performance of the resulting policy (Neumann, 2011).

We will discuss different variants of the Hierarchical REPS (HiREPS) approach with increasing complexity. We start our discussion with the contextual, continuous-armed bandit case where the episode ends after executing all actions belonging to one sub-policy. sub-policies are selected according to a gating policy $\pi(o|\boldsymbol{s})$ and define a distribution over the action given the state for a predetermined time horizon. Subsequently, we show how to extend this framework to the finite horizon setting, where the agent sequences multiple sub-policies to achieve his goal. Finally, we present the infinite horizon formulation of HiREPS, where the agent keeps executing sub-policies until either the task is fulfilled or a random reset occurs.

1.1 Related Work

Policy search (PS) is a class of RL methods which have been shown to fulfill the requirements of robot RL and is widely used in real robot learning. In applications, PS methods are often preferred over other traditional RL methods such as value-based methods, since they do not require the explicit estimation of a value function. Estimating a value function often requires the agent to fill the state and action space with samples, which is infeasible in robot RL. Therefore, most PS methods are commonly local methods and improve only locally on an initial, sub-optimal solution.

Particular properties of PS methods have been highlighted by several papers. Bagnell and Schneider (2003) show their strong convergence guarantees, Sutton et al. (1999a) discuss their application in combination with function approximators and Stone (2001) improved the possibilities of incorporating domain knowledge. Backing up the theoretical strength of policy search methods, several authors have demonstrated impressive application results. Bagnell and Schneider (2001) use policy search methods to autonomously control helicopters, Rosenstein (2001) learns robot weightlifting, Kohl and Stone (2003) demonstrates learning of a quadrupedal walking behavior and Kober et al. (2008) successfully learn the ball in a cup (also known as Kendama) game. Endo et al. (2008) show the application of PS methods on a biped locomotion and Kormushev et al. (2010) PS to learn how to flip a pan-cake with a robot arm.

Important advances in the area of policy search methods have included pair-wise policy comparisons (Strens, 2003), policy gradient methods (Baxter and Bartlett, 2001; Sutton et al., 1999a) and natural policy gradient methods (Bagnell and Schneider, 2003; Peters and Schaal, 2006). More recently, Kober et al. (2008) presented probabilistic policy search approaches based on expectation maximization. Using EM like methods for reinforcement learning was initially pioneered by Dayan and Hinton (1997). Toussaint et al. (2006) showed how an EM like method can be used not only for MDPs but also partially observed MDPs. Deisenroth (2010) developed a model-based policy search method based on probabilistic modeling. Theodorou et al. (2010) showed a policy improvement algorithm inspired by path integrals and Peters et al. (2010) proposed a PS method that limits the loss of information between policy updates. Recently, Levine and Abbeel (2014) and Schulman et al. (2015) have shown how policy search methods can be combined with neural networks to learn complicated tasks in high dimensional domains. As these results show, current methods are well suited to learn single tasks in isolation. Nevertheless, they frequently fail to scale up to more complex motor tasks as they cannot exploit the hierarchical structure inherent to many tasks.

A common way of representing a hierarchical task structure is to use the concept of options. Options are temporally extended actions and were first introduced by Sutton et al. (1999a) as a way of reducing task complexity. They consist of three components, an action-selection policy, an initiation set that defines in which states an option can be initiated and a termination condition which defines the probability of terminating an option in a given state. Typically, options extend the action space of the agent, i.e., the agent can still take the primitive actions which are active for one time step, and, in addition, the agent can choose the options as temporally extended actions. The use of options can significantly improve the learning speed as the options can be used to connect parts of the state space that are otherwise only reachable with a long action sequence (Sutton et al., 1999a). Consequently, the state space becomes easier to explore and, hence, the learning problem gets simplified. Formally, learning with options can be formulated as learning a Semi-Markov Decision Process (SMDP). SMDPs are not fully Markovian, as the selected option does not only depend on the current state, but also on the active option in the previous time step.

Unlike in the simplest setup where the set of options is given and fixed during learning, it is often also desirable to learn useful options for a given task. In this case, the data efficiency for learning the options can be significantly improved by leveraging all information collected during the execution of one option, also denoted as intra-option learning (Sutton et al., 1998). The state transitions generated by a certain option can also be used to update other options in an off-policy learning setup. Levy and Shimkin (2012) proposed a different view of intra-option learning by augmenting the state space with the option-termination probability. In that augmented state space, the option selection policy and termination condition may be represented by orthogonal basis functions and is optimized individually by standard policy gradient methods.

Options are also used in many hierarchical RL approaches where they are often extended to subtasks. A sub-task also contains an individual reward function such that we can learn the policy of the subtask. Dietterich (2000) proposed the MAXQ framework that uses several layers of subtasks. In contrast to the traditional option framework, the policies of the subtasks can also choose to execute a new subtask instead of just a primitive action. The subtasks are given by an individual reward function per subtask, a termination condition and a set of actions or other subtasks that can be executed. The policy of the subtasks is learned by the MAXQ-Q learning algorithm that learns the optimal value function for each subtask. However, the structure of the subtasks, i.e., the individual reward functions, must be specified by the user. How to define such subtasks for complex robot motor tasks or even learn the structure from data remains an open question. Additionally, value function methods are less applicable in continuous state action domains and not well suited for robotic tasks. Barto et al. (2004) used an 'intrinsic motivation' mechanism to define the reward signal of each subtask. A naturally curious agent is exploring its environment and whenever it discovers an unexpected change in its environment, it creates a new subtask in its internal representation that tries to reproduce this unexpected event. Hence, the agent incrementally builds up a set of skills that help the agent to explore its environment more efficiently. After this exploration phase, new tasks can be learned more efficiently by using the learned set of skills.

The notion of subtasks has also been used in continuous environments. Morimoto and Doya (2001) proposed a hierarchical RL setup where the subtask is defined by reaching a specific joint configuration of the robot. In this work, the set of subtasks, i.e., desired joint configurations, is given and the robot learns to reach the individual joint configurations as well as to choose a reachable next desired joint configuration. Ghavamzadeh and Mahadevan (2003) used a policy gradient approach to learn the policies for the individual subtasks while a value based method is used to select the next subtask. Policy gradient approaches work well in continuous environments but often converge slowly. In both approaches, the subtasks have been specified by the user and could not be modified by the learning algorithm. Hence, both approaches are limited to simpler robots where appropriate subtasks can be hand-tuned. In Konidaris and Barto (2009), the structure of the subtasks is also learned by discovering the initiation set, the termination set and the option policy for new options. The options are then be chained together to solve the overall task. However, the option discovery algorithm is to date still limited to rather simple agents, such as a ball navigating in a maze.

The approach which is probably most closely related to ours is Bayesian policy search with hierarchical policy priors (Wingate et al., 2011). In this approach, the probabilistic formulation of policy search (Kober et al., 2008) is used and a hierarchical Bayesian prior is used for the policy parameters. Similarly to our approach, the activations of options are modeled as latent variables in their model that are estimated by MCMC sampling methods. While the use of hierarchical Bayesian priors is a promising idea, the approach is restricted to directly learn on the trajectory data instead of state actions pairs, which might lead to inefficient policy updates. In addition, the approach cannot be extended straightforwardly to complex high dimensional robots as the structure of the options is limited.

Another approach to simplifying the hierarchical RL problem in continuous action spaces is to restrict the space of possible trajectories by using parametrized policies, often also called movement primitives (Schaal et al., 2003), motion templates (Neumann and Peters, 2009) or parametrized skills (Da Silva et al., 2012). Learning the option policy now reduces to learning an appropriate parameter vector for the option. In Neumann and Peters (2009), the agent learned to sequence such parametrized policies, i.e., it learned the correct order of the parametrized policy as well as the single parameter vectors of the policy. In total, the agent has to learn fewer decisions as opposed to directly learning with primitive actions. However, learning is often also more difficult as the effect of an inaccurate decision can be much more costly than for primitive actions. While Neumann and Peters (2009) used value function approximation to evaluate the quality of the chosen parameter vectors of the options, Stulp and Schaal (2012) directly used the reward to come as evaluation. This reward to come was subsequently used by the Policy Improvement by Path Integrals (PI^2) algorithm (Theodorou et al., 2010) to learn the option policy.

Parametrized options or skills were also used for skill transfer, i.e., generalizing the parameters of the options to new tasks (Kupcsik et al., 2014). Thomas and Barto (2012) showed how the structure of motor primitives for continuous state action tasks can be learned and later be used to accelerate learning of a similar task. Da Silva et al. (2012) as well as Kober et al. (2010b) generalized parametrized skills for reasonably similar tasks drawn from a task distribution. They do not only use the concept of an option for a single policy, but instead allow each option to adapt to a subset of the task space by smoothly changing the parameter vector of the option. Hence, individual options are responsible for areas of the task space that are locally smooth. Multiple options together span discontinuous areas of the task space. The proposed framework consists of a pipeline of machine learning tools to identify the individual smooth lower dimensional manifolds as well as to generalize and improve the policy parameters. Alternative approaches for task generalization of parametrized options include learning a mixture of experts (Mülling et al., 2013) and Gaussian processes (Ude et al., 2010). The method presented in this paper is applicable to both primitive actions as well as movement primitives.

In the remainder of the paper we will explain how to obtain at a hierarchical formulation of the relative entropy policy search (REPS) method. We treat the problem of learning a hierarchical policy as a latent variable estimation problem. For the policy update, we assume that we can only observe the resulting actions of the old policy. The underlying hierarchy is unknown and, therefore, unobserved. The resulting algorithm is closely related to expectation maximization (EM) for latent variable models. We prove that such EM mechanisms can be incorporated in the information theoretic regularization of the REPS algorithm in order to get a lower bound of the original optimization problem which can, subsequently, be optimized in closed form. Furthermore, we introduce an additional constraint into the optimization problem that bounds the uncertainty of identifying a sub-policy given an action. As a consequence, the sub-policies are separated in the action space, which results in finding more versatile solutions and also alleviates the problem of averaging over several modes in the solution space, which is present in many current policy search algorithms as shown in Neumann (2011). In the evaluation section, we compare our algorithm to other state-of-the-art methods on benchmark problems and evaluate our algorithm on real world robotic applications.

2. Background on Information Theoretic Policy Search

As our algorithm is based on information theoretic policy search (Peters et al., 2010; Daniel et al., 2012a), we will quickly review the most relevant concepts of information theoretic policy search in the non-hierarchical learning setup. In policy search, an agent tries to maximize the expected return by adapting a parametrized policy. Formally, in a Markov decision process setting, the agent is in a state $\boldsymbol{s} \in \mathbb{S}$ and chooses an action $\boldsymbol{a} \in \mathbb{A}$ to execute. Given state $\boldsymbol{s}$ and action $\boldsymbol{a}$, the agent transfers to a next state $\boldsymbol{s}'$ in accordance with a transition probability distribution $p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) = \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}}$. For each transition, the agent also receives a reward $r \in \mathbb{R}$ that depends on $\boldsymbol{s}$ and $\boldsymbol{a}$, which we also write as $\mathcal{R}_{\boldsymbol{sa}}$. The agent chooses actions $\boldsymbol{a}$ in the current state $\boldsymbol{s}$ according to a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$. We will consider an average reward setting where the goal of the agent is to find an optimal policy that will maximize the average reward

$$J(\pi) = \mathbb{E}[\mathcal{R}_{\boldsymbol{sa}}] = \iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathcal{R}_{\boldsymbol{sa}} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \tag{1}$$

where $\mu^{\pi}(\boldsymbol{s})$ is the state visit distribution of policy π.

Information-theoretic policy search was introduced with the relative entropy policy search (REPS) algorithm (Peters et al., 2010). We will start by defining a constrained optimization problem for solving the discussed average reward reinforcement learning setting and, subsequently, add an information-theoretic constraint to make the optimization problem feasible.

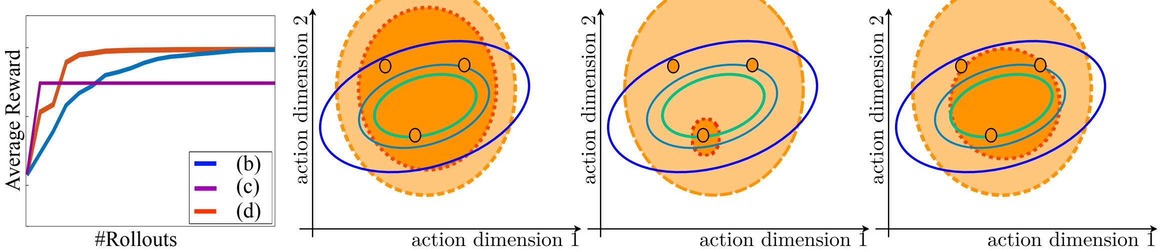
Constraints for the State Distribution. The objective of our optimization problem is given in Equation (1) which is supposed to be maximized with respect to $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. The agent can not freely choose the state distribution $\mu^{\pi}(\boldsymbol{s})$ of the policy, since $\mu^{\pi}(\boldsymbol{s})$ needs to comply with the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ and the system dynamics $\mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}}$, i.e.,

$$\forall \boldsymbol{s}' : \mu^{\pi}(\boldsymbol{s}') = \int_{\boldsymbol{s},\boldsymbol{a}} \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}.$$

However, direct matching of the state probabilities is not feasible in continuous state spaces. Instead, we introduce state features $\boldsymbol{\phi}(\boldsymbol{s})$ and only match the feature averages

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^{\pi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' = \iiint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}}\boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{s}'. \tag{2}$$

For example, if we use all linear and squared terms of $\boldsymbol{s}$ in our feature vector $\boldsymbol{\phi}(\boldsymbol{s})$, we match the mean and the variance under both distributions. Additionally, we require the joint probability



(a) Learning Curves (b) Update not greedy enough (c) Update too greedy (d) Balanced Exploration

Figure 1: Schematic sketch of the behavior of policy updates. Solid lines show the contours of a quadratic reward function. The orange shaded area delimited by the dashed lines illustrates the area within two times the standard deviation of the sampling policy and the orange dots represent samples from the original policy. The darker shaded area delimited by the dotted line indicates the the possible policy update. The policy update needs to be balanced, such that it converges quickly to the local optimum, while not converging too fast to miss it. Such a balance is chosen by the user by specifying the relative entropy bound. Due to the relative entropy bound, the step size of the update is invariant to the task, the reward function or the parametrization of the policy. a) Learning curves corresponding to illustrations b-c. b) The update is not greedy enough, the policy will take too long to converge. c) The update is too greedy, the policy will not find the optimal solution. d) The update balances exploration and exploitation such that the policy quickly converges to a local optimum.

$p(\boldsymbol{s}, \boldsymbol{a}) = \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ to define a probability distribution, i.e., its integral has to evaluate to

$$1 = \iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \tag{3}$$

While the optimization criterion given in Equation (1) with constraints in Equations (2,3) describes the general average reward reinforcement learning problem, it does not consider that we have typically estimated $\mathcal{R}_{\boldsymbol{sa}}$ and $\mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}}$ from a limited amount of data. Hence, we do not want the agent to 'jump' to the optimum of this problem but instead balance exploitation versus exploration. In REPS (Peters et al., 2010), this exploitation-exploration trade off is balanced by the insight that the loss of information in the policy updates should be limited. Independently, such regularization was also suggested and motivated from different perspectives by other authors. Still and Precup (2011) showed that the policy update with maximum information gain results in a similar solution and Azar et al. (2012) motivated a similar update as punishing the distance between the controlled system and the uncontrolled one. Finally, Rawlik et al. (2012) showed that even previous probabilistic policy search approaches are closely related. All these arguments have lead to similar solutions despite their different motivations and the resulting algorithms work well on benchmark problems.

Staying Close to the Data by Information Theoretic Constraints. As motivated by Peters et al. (2010), one key feature of effective PS methods is to limit the loss of information between policy updates, i.e., while we want to maximize the average reward of the new policy, we also want to stay close to the 'data', i.e., the state action distribution $q(\boldsymbol{s}, \boldsymbol{a})$ of the old policy. Staying close to the data can be achieved by limiting the Kullback-Leibler (KL) divergence between the observed data $q(\boldsymbol{s}, \boldsymbol{a})$ and the next policy, i.e.,

$$\epsilon \geq D_{\mathrm{KL}}\left(\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \,||\, q(\boldsymbol{s}, \boldsymbol{a})\right) = \iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \log \frac{\mu(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})}{q(\boldsymbol{s}, \boldsymbol{a})} \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}. \tag{4}$$

Maximizing Equation (1) under the constraints of Equations (2, 3, 4) yields the optimization problem that defines the REPS method (Peters et al., 2010) which is the basis for our hierarchical policy search algorithm. The REPS optimization problem allows for a closed form solution for $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ that can be derived by the method of Lagrangian multipliers. It is given by

$$\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \propto q(\boldsymbol{s}, \boldsymbol{a}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} + \mathbb{E}\left[\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{a}\right] - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right), \tag{5}$$

where η and $\boldsymbol{\theta}$ are Lagrangian parameters that are obtained by optimizing the dual function

$$g(\eta, \boldsymbol{\theta}) = \eta \log \iint q(\boldsymbol{s}, \boldsymbol{a}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})}{\eta}\right) \mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a} + \eta\epsilon, \tag{6}$$

of the original optimization problem. If we interpret the term $V(\boldsymbol{s}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$ as value function linear in the parameters $\boldsymbol{\theta}$, the term $\mathcal{R}_{\boldsymbol{sa}} + \mathbb{E}\left[V(\boldsymbol{s}')\right] - V(\boldsymbol{s})$ can be viewed as advantage function. The Lagrangian parameter η is a scaling factor for the advantage. It becomes the temperature of the soft-max distribution defined in Equation (5) such that the KL-bound from Equation (4) is met.

Sample-Based REPS. As the reward function and the old distribution $q(\boldsymbol{s}, \boldsymbol{a})$ can have an arbitrary structure, the integral contained in the dual function can typically not be obtained in closed form. However, the reward function can typically be evaluated on samples from $q(\boldsymbol{s}, \boldsymbol{a})$ and these samples can be used to approximate the dual function g. As a result, we can only evaluate the distribution $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ on a finite set of samples $\boldsymbol{s}_i$ and $\boldsymbol{a}_i$ with $i = 1 \ldots N$.

In order to sample new actions $\boldsymbol{a}$ in the next iteration of the algorithm, we need to find a parametric representation $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})$ of the policy that approximates the distribution $\pi(\boldsymbol{a}_i|\boldsymbol{s}_i)$, where $\boldsymbol{\beta}$ denotes the parameter vector of the policy. To do so, we aim to minimize the expected Kullback-Leibler divergence between $\pi(\boldsymbol{a}_i|\boldsymbol{s}_i)$ and the next parametric policy $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})$, i.e.,

$$\begin{aligned}
&\arg\min_{\boldsymbol{\beta}} \int p(\boldsymbol{s}) D_{\mathrm{KL}}\left(\pi(\boldsymbol{a}|\boldsymbol{s}) \,||\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})\ \right)\,\mathrm{d}\boldsymbol{s} \\
&= \arg\min_{\boldsymbol{\beta}} \int p(\boldsymbol{s}) \int \pi(\boldsymbol{a}|\boldsymbol{s}) \log \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})} \,\mathrm{d}\boldsymbol{a}\,\mathrm{d}\boldsymbol{s} \\
&\approx \arg\max_{\boldsymbol{\beta}} \sum_i \frac{p(\boldsymbol{s}_i, \boldsymbol{a}_i)}{q(\boldsymbol{s}_i, \boldsymbol{a}_i)} \log \tilde{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i; \boldsymbol{\beta}) + \mathrm{const} \\
&= \arg\max_{\boldsymbol{\beta}} \sum_i w_i \log \tilde{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i; \boldsymbol{\beta}),
\end{aligned} \tag{7}$$

with

$$w_i = \exp\left(\frac{r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \mathbb{E}\left[\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}')\middle|\, \boldsymbol{s}_i, \boldsymbol{a}_i\right] - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}_i)}{\eta}\right). \tag{8}$$

Since the available samples are drawn from the distribution $q(\boldsymbol{s}, \boldsymbol{a})$, and not $p(\boldsymbol{s}, \boldsymbol{a})$, we need to perform importance sampling and divide by the sampling distribution $q(\boldsymbol{s}, \boldsymbol{a})$. Equation (7) is equivalent to computing the weighted maximum likelihood estimator for $\boldsymbol{\beta}$. In Section 4.3 we discuss how to compute the expectation over next states $\mathbb{E}\left[\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}')\middle|\, \boldsymbol{s}_i, \boldsymbol{a}_i\right]$.

Illustration of Information Theoretic Policy Search. We illustrate the concept of information theoretic policy search on a two-dimensional toy problem with a quadratic reward function in Figure 1(a). Here, the algorithm typically starts with a broad explorative policy. Given a set of samples generated using this policy, we update the policy such that the exploitation-exploration trade-off is well balanced. This trade-off is chosen by the relative entropy bound of the REPS algorithm. Due to the relative entropy bound, the step-size of the policy update is largely independent of the task, the reward function, or the representation of the policy. Changing the policy only by a small step leads to further exploration as we stay close to the old exploration policy as illustrated in Figure 1(b). In contrast, a large KL divergence for the policy update greedily jumps to the best observed samples with very little further exploration, see Figure 1(c). This trade-off has to be chosen by the user by specifying the bound ϵ for the KL-divergence. Typically, we want to achieve a moderate exploration-exploitation trade-off as illustrated in Figure 1(d).

3. Learning Hierarchical Policies for Real Robot Reinforcement Learning

In this section, we will extend the REPS optimization problem to the hierarchical case, where the agent learns a hierarchical policy that is based on both a gating network and sub-policies. In order to obtain an efficient update rule for the hierarchical policy, we will rely on several insights detailed below. We will use these insights to derive three different learning algorithms that can be employed in different scenarios.

Hierarchical Policies. More complex tasks often require multiple sub-policies, such as a forehand stroke, backhand stroke, smash or lob in tennis. Different sub-policies are appropriate in different states of the environment. In order to model a policy consisting of several sub-policies, we use a hierarchical policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ which consists of a set of sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and a gating network $\pi(o|\boldsymbol{s})$ that selects the currently active sub-policy. Thus, the hierarchical policy can be represented as

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \sum_{o \in \mathcal{O}} \pi(o|\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}, o). \tag{9}$$

In order to determine the action $\boldsymbol{a}$ in state $\boldsymbol{s}$, we first sample a sub-policy from the gating network $\pi(o|\boldsymbol{s})$ and, subsequently, sample the action $\boldsymbol{a}$ from the specific sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$. The benefit of representing multiple solutions has already been made evident in recent research results (Calinon et al., 2013; Daniel et al., 2012a).

Options as Latent Variables. Sharing of experience between sub-policies, known as inter-option learning, is an important property of a hierarchical learning algorithm for data-efficient learning. To realize inter-option learning, we treat the problem of learning sub-policies as a latent variable problem, i.e., we assume to observe only state-action samples $\{\boldsymbol{s}, \boldsymbol{a}\}$, but not the index of the generating sub-policy o. Instead, we infer the latent structure of the hierarchical policy that has generated the re-weighted samples. Expectation-maximization based methods can be used to iteratively estimate the responsibilities $p(o|\boldsymbol{s}, \boldsymbol{a})$ of the sub-policies for each sample $\{\boldsymbol{s}, \boldsymbol{a}\}$ and, subsequently, update

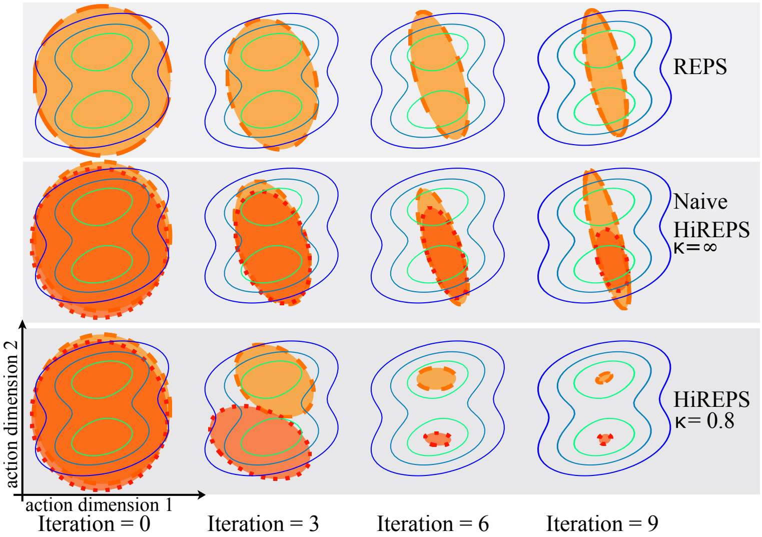


Figure 2: Schematic sketch of the behavior of REPS, and HiREPS with ($\tilde{\kappa} = 0.8$) and without ($\tilde{\kappa} = \infty$) bounding of the gating's entropy. A two-dimensional, bi-modal reward function is shown in the contour plot lines. Two sub-policies are shown in dashed and dotted outlines respectively (outlines show 95% confidence boundaries). REPS and naive HiREPS average over both modes. Constrained HiREPS separates the sub-policies and is therefore able to find both modes.

the sub-policies where the influence of each sample $\{\boldsymbol{s}, \boldsymbol{a}\}$ for the update of the sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ is weighted by the responsibility $p(o_j|\boldsymbol{s}, \boldsymbol{a})$. Hence, the generated experience can be shared between the sub-policies. In Section 3.1, we integrate such an expectation-maximization mechanism into the REPS algorithm.

Learning Versatile Solutions. Being able to represent multiple solutions does not force the learning algorithm to find different solutions. Thus, without further constraints, we are likely to find multiple sub-policies that concentrate on the same mode of the solution space. Versatility of sub-policies can be achieved if the sub-policies are clearly separated in the state-action space. To enforce this separation of the sub-policies, we limit the expected change in the entropy H of the responsibilities of the sub-policies, i.e.,

$$\kappa \geq \frac{\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}[H\big(p(o|\boldsymbol{s},\boldsymbol{a})\big)]}{\mathbb{E}_{q(\boldsymbol{s},\boldsymbol{a})}[H\big(q(o|\boldsymbol{s},\boldsymbol{a})\big)]} = \frac{\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{o \in \mathcal{O}} p(o|\boldsymbol{s},\boldsymbol{a}) \log p(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}}{\iint q(\boldsymbol{s},\boldsymbol{a}) \sum_{o \in \mathcal{O}} q(o|\boldsymbol{s},\boldsymbol{a}) \log q(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}}, \tag{10}$$

where $\mathbb{E}_{q(\boldsymbol{s},\boldsymbol{a})}[H\big(q(o|\boldsymbol{s},\boldsymbol{a})\big)]$ is a constant and we write $\tilde{\kappa} = \mathbb{E}_{q(\boldsymbol{s},\boldsymbol{a})}[H\big(q(o|\boldsymbol{s},\boldsymbol{a})\big)]\kappa$ to simplify the notation, such that the constraint reads

$$\tilde{\kappa} \geq \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}[H\big(p(o|\boldsymbol{s},\boldsymbol{a})\big)] = -\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{o \in \mathcal{O}} p(o|\boldsymbol{s},\boldsymbol{a}) \log p(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \tag{11}$$

A high entropy of the responsibility $p(o|\boldsymbol{s}, \boldsymbol{a})$ indicates a high uncertainty in deciding which sub-policy has generated the state action pair, which implies that several sub-policies do overlap in the parameter space. By limiting the entropy, such overlapping is avoided and we ensure that different sub-policies concentrate on different and separate solutions.

We will discuss three settings for Hierarchical REPS (HiREPS). In the episodic setup, a single sub-policy is executed per episode, but the algorithm can choose between multiple sub-policies and adapt these to the current context. Subsequently, we will show how to sequence a fixed number of sub-policies with a finite horizon MDP formulation. Finally, we will use a infinite horizon MDP formulation to learn how to sequence a possibly infinite number of skills.

Illustration of Hierarchical Learning. We illustrate the advantage of learning with multiple sub-policies on a toy task with a two dimensional action space and a bi-modal reward function (see Figure 2). In this task, the reward function consists of two attractors $\boldsymbol{g}_1, \boldsymbol{g}_2$ and a reward scaling matrix Σ_r, such that the reward function is given by

$$r(\boldsymbol{a}) = -\min_i \left((\boldsymbol{a} - \boldsymbol{g}_i) \Sigma_r (\boldsymbol{a} - \boldsymbol{g}_i)^T \right),$$

and has two local optima at $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$, respectively. The illustration in Figure 2 demonstrates two key insights into PS methods. First, many standard policy search methods such as Policy Improvement with Path Integrals (Theodorou et al., 2010) or EM-based policy search methods (Kober et al., 2008) will be attracted to multiple optima in a multi-modal solution space and, therefore, converge slowly due to averaging over several modes (Neumann, 2011). Second, we would like to represent a versatile solution space by learning all modes of the reward function. Figure 2 shows a qualitative comparison of HiREPS to the standard REPS algorithm which can only use one sub-policy and to the naive implementation of HiREPS that does not bound the sub-policies' entropy (i.e., $\tilde{\kappa} = \infty$). The single sub-policy algorithm tries to average over both modes and, takes a long time to converge. The naive HiREPS exhibits similar behavior. Both sub-policies are attracted by both modes. In most cases, both sub-policies will find the same mode and convergence will be slower. Thus, only introducing hierarchical policies without additional constraints cannot take full advantage of the increased flexibility. When limiting the entropy, however, the sub-policies quickly separate and concentrate on the two individual modes, allowing for a fast improvement of the policy without getting stuck between two modes.

3.1 Episodic Selection of Sub-Policies

We start our discussion of HiREPS with the continuous multi-armed contextual bandit setting. In this setting, the agent is presented with an initial state according to an initial state distribution $p^0(\boldsymbol{s})$ and has to select a sub-policy o as well as an action $\boldsymbol{a}$ according to this state. In this setting, an episode consists of executing exactly one sub-policy, afterwards the episode is terminated and the environment is reset. Thus, no state transition is modeled, as the whole episode consists of only one step, i.e., executing one sub-policy until it terminates.

Contextual Optimization of the Policy. While we do not need to use the state distribution constraint from REPS, as we do not have to incorporate state transitions, the agent can still not freely choose its state action distribution $p(\boldsymbol{s}, \boldsymbol{a}) = \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ as the initial state distribution $p^0(\boldsymbol{s})$ is specified by the learning task. Hence, the estimated state distribution $\mu^\pi(\boldsymbol{s})$ has to satisfy $\mu^\pi(\boldsymbol{s}) = p^0(\boldsymbol{s})$ for all $\boldsymbol{s}$. As this requirement would result in an infinite number of constraints, we need to resort to matching feature averages, i.e.,

$$\hat{\boldsymbol{\phi}} = \sum_{o \in \mathcal{O}} \iint \mu^\pi(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \pi(o|\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \tag{12}$$

where $\hat{\boldsymbol{\phi}}$ denotes the average observed feature vector for the initial state

$$\hat{\boldsymbol{\phi}} = \iint q(\boldsymbol{s}, \boldsymbol{a}) \boldsymbol{\phi}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \tag{13}$$

This constraint enforces that the learned state action distribution does not concentrate only on contexts where the task is easy to achieve, but considers all contexts sampled from the initial state distribution.

Resulting Optimization Problem. For the resulting optimization problem, we combine the insights from the previous section on hierarchical policies with the contextual episodic learning constraint in Eq. (12). The episodic learning problem of multiple sub-policies is then given as

$$\begin{aligned}\max_{\pi,\mu} J(\pi) = \max_{\pi,\mu} &\sum \iint \mu^{\pi}(\boldsymbol{s})\pi(o|\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\mathcal{R}_{\boldsymbol{sa}}\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},\\
\text{s. t.}\quad & \epsilon \geq D_{\mathrm{KL}}\left(\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})||\, q(\boldsymbol{s},\boldsymbol{a})p(o|\boldsymbol{s},\boldsymbol{a})\right),\\
& \tilde{\kappa} \geq \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}\left[H\big(p(o|\boldsymbol{s},\boldsymbol{a})\big)\right],\\
& \hat{\boldsymbol{\phi}} = \sum_{o\in\mathcal{O}}\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},\\
& 1 = \sum \iint \mu^{\pi}(\boldsymbol{s})\pi(o|\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)d\boldsymbol{s}d\boldsymbol{a}.\end{aligned} \tag{14}$$

In this optimization problem, the responsibilities

$$p(o|\boldsymbol{s},\boldsymbol{a}) = \frac{\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})}{\sum_{o\in\mathcal{O}}\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})},$$

occur inside the log-term of the KL-divergence. As the responsibilities contain a sum in the denominator, we also obtain the sum inside the log-term, preventing us from solving for $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})$ in closed form. However, we can resort to an iterative, expectation maximization (EM) update strategy shown in Daniel et al. (2012a). In the expectation step, we first fix the sub-policies and compute the responsibilities $\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})$. In the maximization step, we replace $p(o|\boldsymbol{s},\boldsymbol{a})$ in our optimization problem with the pre-computed responsibilities $\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})$ and, therefore, neglect that the responsibilities will change once we change the sub-policies. In Appendix A we show that the EM-step maximizes a lower bound of the original optimization problem. We use this iterative update strategy in all further algorithms. The adapted optimization problem reads as

$$\begin{aligned}\max_{\pi,\mu} J(\pi) = \max_{\pi,\mu} &\,\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},o}\left[\mathcal{R}_{\boldsymbol{sa}}\right],\\
\text{s. t.}\quad & \epsilon \geq D_{\mathrm{KL}}\left(\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})||\, q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\right),\\
& \tilde{\kappa} \geq -\sum_{o\in\mathcal{O}}\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\log\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},\\
& \hat{\boldsymbol{\phi}} = \sum_{o\in\mathcal{O}}\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},\\
& 1 = \sum_{o\in\mathcal{O}}\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}.\end{aligned} \tag{15}$$

The above optimization problem can be solved by the method of Lagrange multipliers. The Lagrangian also allows for a closed form of $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})$ which is given as

$$\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \propto q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\frac{\xi}{\eta}}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} - V(\boldsymbol{s})}{\eta}\right), \tag{16}$$

which depends on the Lagrangian parameters ξ, η and $\boldsymbol{\theta}$ with $V(\boldsymbol{s}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s})$. The Lagrangian parameter ξ is associated to the overlapping constraint in Eq. (11). In this update equation, the parameter ξ controls the spreading of the sub-policies. A higher value of ξ will force the sub-policies to spread further apart and reduce overlapping of sub-policies as the influence of state-action pairs with a high entropy of $p(o|\boldsymbol{s}, \boldsymbol{a})$ is weakened. The derivation of the dual formulation is given in the appendix.

In Table 3, we give the algorithmic form of episodic HiREPS. The new parametric policy $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}, o; \boldsymbol{\beta}_o)$ has to be computed from the weighted samples by performing a weighted maximum likelihood estimate for the single sub-policy parameters $\boldsymbol{\beta}_o$ as well as the parameter vector for the gating policy. In HiREPS, we obtain a weight

$$w_{i,o} = \tilde{p}(o|\boldsymbol{s}_i, \boldsymbol{a}_i)^{1+\frac{\xi}{\eta}} \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} - V(\boldsymbol{s}_i)}{\eta}\right),$$

for each sample and each sub-policy. These weights can be used to update the policy. HiREPS does not make any assumptions on the form of the policy and many different representations could be considered. In Section 4, we show the details of the policy model we chose to implement for the evaluations, i.e., a soft-max gating policy and linear Gaussian sub-policies.

Since HiREPS does not assume knowledge of which sub-policy generated a sample, it can use all samples to update all sub-policies, also known as inter-option learning. The weights $w_{i,o}$ are then used to update the sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and the gating policy $\pi(o|\boldsymbol{s})$.

3.2 Sequencing of Skills

Many real world tasks require not only one, but multiple sequential interactions. For example, in a game of tennis, we need to perform several tennis strokes in sequence in order to win the game. Just learning to return a ball is insufficient to win a game. Instead, the ball has to returned in such a way that future strokes can lead to situations where the opponent is unable to return the ball. Hence, tasks like tennis can be learned more efficiently if we learn a sequence of tennis strokes against an opponent. We will model the skill sequencing case with a limited number K of sequential decisions. We will denote the individual action selection problems as stages of the decision problem. Our goal is to maximize the sum of the expected reward over all stages, i.e.,

$$\begin{aligned} J &= \mathbb{E}_{\boldsymbol{a}_{1:K}, \boldsymbol{s}_{1:K+1}}\left[r_{K+1}(\boldsymbol{s}_{K+1}) + \sum\nolimits_{k=1}^{K} r_k(\boldsymbol{s}_k, \boldsymbol{a}_k)\right] \\ &= \int \mu^{\pi}_{K+1}(\boldsymbol{s}) r_{K+1}(\boldsymbol{s}) d\boldsymbol{s} + \iint \sum_{k=1}^{K} \mu^{\pi}_k(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}) r_k(\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \end{aligned} \tag{17}$$

where $\mu^{\pi}_k(\boldsymbol{s})$ are the state distributions at for each decision step. The function $r_{K+1}(\boldsymbol{s}_{K+1})$ denotes the final reward for reaching state $\boldsymbol{s}_{K+1}$ and $r_k(\boldsymbol{s}_k, \boldsymbol{a}_k)$ denotes the reward for executing an action $\boldsymbol{a}_k$ in state $\boldsymbol{s}_k$. In the tennis example, the individual rewards could, for example, describe the energy efficiency of the movements, while the overall reward signal carries information about winning or losing the point. The sub-policy is given by

$$\pi_k(\boldsymbol{a}|\boldsymbol{s}) = \sum_{o \in \mathcal{O}} \pi_k(o|\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o). \tag{18}$$

```
Input: Information loss tolerance $\epsilon$, Entropy tolerance $\tilde{\kappa}$, Number of sub-policies $O$, number of Iterations $L$, number of episodes per iteration $M$.
Initialize all $\pi(\boldsymbol{a}|\boldsymbol{s},o)$ and $\pi(o|\boldsymbol{s})$.
for $l = 1$ to $L$   ...   # iterations
    Collect samples
        for $i = 1,\dots,M$        (# episodes)
            Sample initial state $\boldsymbol{s}_{1,i}$ from environment.
                Sample action: $\boldsymbol{a}_i \sim q(\boldsymbol{a}|\boldsymbol{s}_i) = \sum_{o\in\mathcal{O}} \pi_{\text{old}}(o|\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{a}|\boldsymbol{s}_i,o)$.
                Execute action $\boldsymbol{a}_i$ and observe reward $r(\boldsymbol{a}_i,\boldsymbol{s}_i)$.

    Compute Responsibilities:
        $\tilde{p}(o|\boldsymbol{s}_i,\boldsymbol{a}_i) = p_{\text{old}}(o|\boldsymbol{s}_i,\boldsymbol{a}_i) = \frac{\mu^{\pi}_{\text{old}}(\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{a}_i|\boldsymbol{s}_i o_i)\pi_{\text{old}}(o|\boldsymbol{s}_i)}{\sum_{o\in\mathcal{O}}\mu^{\pi}_{\text{old}}(\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{a}_i|\boldsymbol{s}_i o_i)\pi_{\text{old}}(o|\boldsymbol{s}_i)}$ for all $i$.

    Minimize the dual function
        $[\boldsymbol{\theta}^*,\eta^*,\xi^*] = \arg\min_{[\boldsymbol{\theta},\eta,\xi]} g\left(\boldsymbol{\theta},\eta,\xi\right)$.

    Policy update:
        Compute model distribution
            $\mu^{\pi}(\boldsymbol{s}_i)\pi(\boldsymbol{a}_i|\boldsymbol{s}_i o_i)\pi(o|\boldsymbol{s}_i) \propto \tilde{p}(o|\boldsymbol{s}_i,\boldsymbol{a}_i)^{1+\xi^*/\eta^*}\exp\left(\frac{R_i - V^*(\boldsymbol{s}_i)}{\eta^*}\right)$.
        Estimate policies
            $\pi(o|\boldsymbol{s})$ and $\pi(\boldsymbol{a}|\boldsymbol{s},o)$ for all $o = 1\dots O$ by weighted ML estimates.

Output: Policies $\pi(\boldsymbol{a},o|\boldsymbol{s})$
```

Table 1: Episodic HiREPS. In each iteration the algorithm starts by sampling an sub-policy o from the gating policy $\pi(a|\boldsymbol{s})$ given the initial state $\boldsymbol{s}$ and an action $\boldsymbol{a}$ from $\pi(\boldsymbol{a}|\boldsymbol{s},o)$ from the sub-policy. Subsequently, the action is executed to generate the reward $r(\boldsymbol{s},\boldsymbol{a})$. The parameters η, ξ and $\boldsymbol{\theta}$ are determined by minimizing the dual-function g.

Alternatively, we can also share the sub-policies for all decision stages and only make the gating policy time dependent, i.e.,

$$\pi_k(\boldsymbol{a}|\boldsymbol{s}) = \sum_{o\in\mathcal{O}} \pi_k(o|\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o).$$

This formulation allows for reusing experience between the stages. Whether this property is useful depends on the task at hand. For example in tennis, the strokes at the decision stages can taken from the same skill library[1]. In the case of skill sequencing, we have one latent variable $o_{i,k}$ per rollout i and per decision stage k.

Connecting the Decision Stages. The state distributions $\mu^{\pi}_k(\boldsymbol{s})$ are now connected through the transition dynamics $\mathcal{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}$ yielding

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^{\pi}_{k+1}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' = \sum_{o\in\mathcal{O}} \iiint \mathcal{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}\mu^{\pi}_k(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a}. \qquad (19)$$

[1] Except for the first stroke, as it is supposed to be the serve.

In addition, the agent cannot freely choose the initial state distribution $\mu_1^\pi(\boldsymbol{s})$, but needs to match the given initial state distribution $p_1(\boldsymbol{s})$, which is implemented in the same way as for the episodic case. Typically the dynamics $\mathcal{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{a}}$ of a task are not known and need to be estimated from data.

Skill Sequencing Optimization Problem. After including the overlapping constraints from the previous section, the optimization problem reads

$$\begin{aligned}
&\max_{\pi_k,\mu_k^\pi} \ \mathbb{E}_{\boldsymbol{a}_{1:K},\boldsymbol{s}_{1:K+1}}\left[r_{K+1}(\boldsymbol{s}_{K+1}) + \sum\nolimits_{k=1}^{K} r_k(\boldsymbol{s}_k,\boldsymbol{a}_k)\right],\\
&\text{s.t.} \quad \epsilon \geq D_{\mathrm{KL}}\left(\mu_{K+1}^\pi(\boldsymbol{s})||\, q_{K+1}(\boldsymbol{s})\right),\\
&\hat{\boldsymbol{\phi}}_1 = \int \boldsymbol{\phi}(\boldsymbol{s}')\mu_1^\pi(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}',\\
&\forall k \leq K : \epsilon \geq D_{\mathrm{KL}}\left(\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})||\, q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})\right),\\
&\tilde{\kappa} \geq -\sum_{o\in\mathcal{O}}\iint \mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})\log\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},\\
&\int \boldsymbol{\phi}(\boldsymbol{s}')\mu_{k+1}^\pi(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' = \sum_{o\in\mathcal{O}}\iiint \mathcal{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{a}}\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a},\\
&1 = \sum\nolimits_{\boldsymbol{s},\boldsymbol{a},o}\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s}).
\end{aligned} \tag{20}$$

The resulting policy update rules are given by

$$\begin{aligned}
&\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s}) \propto\\
&q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\frac{\xi_k}{\eta_k}}\exp\left(\frac{r_k(\boldsymbol{s},\boldsymbol{a}) + \mathbb{E}[V_{k+1}(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}] - V_k(\boldsymbol{s})}{\eta_k}\right),
\end{aligned} \tag{21}$$

where one set of Lagrangian parameters $\xi_k, \eta_t, \boldsymbol{\theta}_k$ is computed for each stage. As we observe, we now obtain an individual value function $V_k(\boldsymbol{s}_k)$ for each decision stage. As in the standard REPS algorithm, the value functions are connected by the advantage function term that occurs inside the exponent in the policy. The skill sequencing algorithm is given in Table 2. The derivations of the dual function and the update rules are given in Appendix A.2. In Section 4.3, we discuss how to compute the expectation over next states $\mathbb{E}[V_{k+1}(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}]$.

3.3 Infinite Horizon

An infinite horizon setting is needed for all repetitive tasks where we sequence an unknown, possibly infinite amount of sub-policies to fulfill the task. For example, when bouncing a ball on a paddle, the repetitive paddling movements induce a clear structure in the motion that suggests the use of sub-policies. In addition, we want to bounce the ball on the paddle for an infinite amount of time. In this setting, the agent needs to consider the state of the environment before each stroke.

```
Input: Information loss tolerance $\epsilon$, entropy tolerance $\tilde{\kappa}$, number of sub-policies $n$, number of time steps $K$, number of iterations $L$.
Initialize all $\pi_k(\boldsymbol{a}|\boldsymbol{s}, o)$ and $\pi_k(o|\boldsymbol{s})$.
for $l = 1$ to $L$ ... # iterations
    Collect samples
        for $i = 1, \ldots, M$ (# episodes)
            Sample initial state $\boldsymbol{s}_{1,i}$ from environment.
            for $k = 1, \ldots, K$ (# motor primitives)
                Sample action: $\boldsymbol{a}_{k,i} \sim q_k(\boldsymbol{a}|\boldsymbol{s}_{k,i}) = \sum_{o \in \mathcal{O}} \pi_{k,\text{old}}(o|\boldsymbol{s}_{k,i})\pi_{k,\text{old}}(\boldsymbol{a}|\boldsymbol{s}_{k,i}, o)$.
                Execute action $\boldsymbol{a}_{k,i}$, observe next state $\boldsymbol{s}_{k+1,i}$ and reward $r(\boldsymbol{a}_{k,i}, \boldsymbol{s}_{k,i})$.
            Observe Final Reward: $r(\boldsymbol{s}_{K+1,i})$.

    Compute Responsibilities:
        $\tilde{p}_k(o|\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i}) = p_{k,\text{old}}(o|\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i})$ for all $k$ and $i$.

    Minimize the dual function
        $[\boldsymbol{\theta}_{1:K+1}{}^*, \eta_{1:K+1}{}^*, \xi_{1:K+1}{}^*] = \arg\min_{[\boldsymbol{\theta}_{1:K+1}, \eta_{1:K+1}, \xi_{1:K+1}]} g\left(\boldsymbol{\theta}_{1:K+1}, \eta_{1:K+1}, \xi_{1:K+1}\right)$.

    Policy update:
        for $k = 1, \ldots, K$
            Compute model distribution
                $\mu_k^\pi(\boldsymbol{s}_{k,i})\pi_k(\boldsymbol{a}_{k,i}|\boldsymbol{s}_{k,i}, o)\pi_k(o|\boldsymbol{s}_{k,i}) \propto$
                $\tilde{p}_k(o|\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i})^{1+\xi_k{}^*/\eta_k{}^*} \exp\left(\frac{R_{k,i}+\mathbb{E}[V^*_{k+1}(\boldsymbol{s}')]-V^*_k(\boldsymbol{s}_{k,i})}{\eta_k{}^*}\right)$.
            Estimate policies
                $\pi_k(o|\boldsymbol{s})$ and $\pi_k(\boldsymbol{a}|\boldsymbol{s}, o)$ by weighted ML estimates.

Output: Policies $\pi_k(\boldsymbol{a}, o|\boldsymbol{s})$ for all $k = 1, \ldots, K$
```

Table 2: Time-indexed HiREPS. In each iteration the algorithm starts by sampling from the policy π_1 given the initial state $\boldsymbol{s}_1$ and executes the sampled action to generate the next state $\boldsymbol{s}_2$. From this state, the next action is sampled with policy π_2. This procedure is repeated until the final time-step is reached. The algorithm observes state transitions and rewards for each step k and the final reward signal $r(\boldsymbol{s})$. The parameters $\eta_{1:K+1}$, $\xi_{1:K+1}$ and $\boldsymbol{\theta}_{1:K+1}$ are determined by minimizing the dual-function g, where $\eta_{1:K+1}$ and $\xi_{1:K+1}$ are vectors containing the Lagrangian parameters η_k and ξ_k for each decision step.

The traditional objective in an infinite horizon MDP is to use the discounted accumulated future rewards, i.e.,

$$R = \sum_{t=0}^{\infty} \gamma^t r_t,$$

where $\gamma < 1$ is a discount factor. Such an objective can be easily transferred to the average reward setting by introducing a reset probability γ, where the agent jumps to a state sampled from the initial state distribution $\mu^0(\boldsymbol{s})$ with probability γ and transitions to the next state with probability $(1 - \gamma)$

(van Hoof et al., 2015). Thus, we obtain a transformed transition probability distribution given by

$$\tilde{p}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) = \gamma p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) + (1-\gamma)\mu^0(\boldsymbol{s}), \tag{22}$$

where γ is the reset probability. Thus, the discount factor is considered as a termination probability. The constraint ensuring the state transition probabilities are satisfied reads as

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^\pi(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' = \sum_{o\in\mathcal{O}} \iiint \tilde{\mathcal{P}}^{\boldsymbol{a}}_{\boldsymbol{ss}'}\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a}. \tag{23}$$

The optimization problem for the infinite horizon case reads as

$$\begin{aligned}
\max_{\pi,\mu^\pi} J(\pi) &= \max_{\pi,\mu^\pi} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},o}\left[\mathcal{R}_{\boldsymbol{sa}}\right], \\
\text{s. t.} \quad \epsilon &\geq D_{\text{KL}}\left(\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})||\, q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\right), \\
\tilde{\kappa} &\geq -\sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\log\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}, \\
\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^\pi(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' &= \sum_{o\in\mathcal{O}} \iiint \gamma\mathcal{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}') + (1-\gamma)\mu^0(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a}, \\
1 &= \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},
\end{aligned} \tag{24}$$

with the resulting policy update

$$\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \propto q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} + \mathbb{E}[V(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}] - V(\boldsymbol{s})}{\eta}\right), \tag{25}$$

where

$$\mathbb{E}[V(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}] = \boldsymbol{\theta}^T\left[\int \gamma\mathcal{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' + \int (1-\gamma)\mu^0(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'\right].$$

In contrast to the finite horizon case, the value function, the policy and the state distributions are now stationary instead of time dependent.

4. Algorithmic Design Choices

Having shown the theoretical foundation of HiREPS, we use this Section to explain implementation details concerning the policy representation and parametrizations as well as feature representations and model learning. The choices detailed in this Section represent only some of the possible implementations and are design choices. The HiREPS framework as specified in the previous Section is independent of these design choices and can be used with arbitrary representations for the gating, the sub-policies as well as arbitrary feature representations. HiREPS only requires that the policy representation can be updated using weighted samples.

```
Input: Information loss tolerance $\epsilon$, Entropy tolerance $\tilde{\kappa}$, Number of sub-policies $O$, number of Iterations $L$, number of rollouts per iteration $M$, reset probability $\gamma$.
Initialize all $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and $\pi(o|\boldsymbol{s})$.
for $l = 1$ to $L$   ...   # iterations
    Collect samples
      for $i = 1, \dots, M$        (# rollouts)
          t = 1; reset = 0;
          Sample initial state $\boldsymbol{s}_{i,t}$ from environment.
          while reset $< \gamma$
              Sample action $\boldsymbol{a}_{i,t} \sim q(\boldsymbol{a}|\boldsymbol{s}_{i,t}) = \sum_{o \in \mathcal{O}} \pi_{\text{old}}(o|\boldsymbol{s}_{i,t}) \pi_{\text{old}}(\boldsymbol{a}|\boldsymbol{s}_{i,t}, o)$.
              Execute  action $\boldsymbol{a}_{i,t}$ and observe reward $r_{i,t}(\boldsymbol{a}_{i,t}, \boldsymbol{s}_{i,t})$.
              Observe reward $r_{i,t}(\boldsymbol{a}_{i,t}, \boldsymbol{s}_{i,t})$ and next state $\boldsymbol{s}_{i,t+1}$.
              Sample reset value.

    Compute Responsibilities:
      $\tilde{p}(o|\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}) = p_{\text{old}}(o|\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}) = \frac{p_{\text{old}}(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}, o)}{\sum_{o \in \mathcal{O}} p_{\text{old}}(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}, o)}$ for all $i, t$.

    Minimize the dual function
      $[\boldsymbol{\theta}^*, \eta^*, \xi^*] = \arg\min_{[\boldsymbol{\theta}, \eta, \xi]} g(\boldsymbol{\theta}, \eta, \xi)$.

    Policy update:
        Compute model distribution
          $p(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}, o) \propto \tilde{p}(o|\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t})^{1+\xi^*/\eta^*} \exp\left(\frac{R_{i,t} + \mathbb{E}[V(\boldsymbol{s}')] - V^*(\boldsymbol{s}_{i,t})}{\eta^*}\right)$.
        Estimate policies
          $\pi(o|\boldsymbol{s})$ and $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ for all $o = 1 \dots O$ by weighted ML estimates.

Output: Policies $\pi(\boldsymbol{a}, o|\boldsymbol{s})$
```

Table 3: Infinite Horizon HiREPS. The algorithm follows the form of the episodic implementation, however one iteration produces state, action, reward triples for each time step in each rollout of an iteration. In the infinite horizon case, a model $\mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}}$ is required to compute $\mathbb{E}[V(\boldsymbol{s}')]$.

4.1 Policy Representation

We implemented the hierarchical policy using a soft-max gating network $\pi(o|\boldsymbol{s})$ and linear Gaussian sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$. The gating network $\pi(o|\boldsymbol{s})$ chooses which option to activate according to the model

$$\pi(o = k|\boldsymbol{s}) = \frac{\exp\left(\boldsymbol{\phi}_G(\boldsymbol{s})^T \boldsymbol{\beta}_{G,k}\right)}{\sum_i^O \exp\left(\boldsymbol{\phi}_G(\boldsymbol{s})^T \boldsymbol{\beta}_{G,i}\right)},$$

where $\boldsymbol{\beta}_{G,1 \dots O}$ are the parameter vectors defining the influence of the sub-policies and $\boldsymbol{\phi}_G$ are the feature representations of the state used for the gating network. While this model itself is relatively simple, the complexity and expressiveness of the resulting policy depends largely on the feature transformations employed for the individual components of the hierarchical policy. In HiREPS, we are free to choose different feature transformations for the sub-policies, the gating and the value function, where the value function features are the features in the primal optimization problem. In the presented experiments, the sub-policies themselves are represented by linear Gaussian policies, i.e.,

$$\pi(\boldsymbol{a}|\boldsymbol{s}, o; \boldsymbol{\beta}_o) = \mathcal{N}\left(\boldsymbol{a}|\boldsymbol{\mu}_o + \boldsymbol{F}_o\boldsymbol{s}, \boldsymbol{\Sigma}_o\right),$$

where $\boldsymbol{\beta}_o = [\boldsymbol{\mu}_o, \boldsymbol{F}_o, \boldsymbol{\Sigma}_o]$ is the parameter tuple describing sub-policy o. The sub-policies could also be defined on a feature transformation of the state. Equally well, non-linear sub-policies instead of the linear Gaussians could be used. However, since HiREPS performs a piecewise linear approximation in the state space using multiple sub-policies, linear sub-policies are usually sufficient. Both, the gating policy as well as the sub-policies can easily be updated using the sample weights computed by HiREPS. The update equations, as shown in the appendix, yield a weight matrix with one row per sample and cone column per sub-policy. The gating requires the full weight matrix as well as the responsibilities to be updated whereas each sub-policy is updated independently using the respective column of the weight matrix to weigh all state-action samples.

Generally, HiREPS can be initialized with many more sub-policies than solutions are expected to be available for the problem at hand. The gating network $\pi(o|\boldsymbol{s})$ will reduce the probability of selecting options that are not concentrated on good solutions to zero and effectively ignore these unnecessary sub-policies. Options can also be pruned throughout the learning process as in the first series of experiments described in Section 5.1.2. There, we report the number of actual solutions found. In this case, whenever the prior $p(o) = \int \pi(o|\boldsymbol{s})\mu(\boldsymbol{s})d\boldsymbol{s}$ of sub-policy activation gets too small (i.e., $p(o) < 10^{-4}$) we delete the sub-policy. However, in order to avoid sub-policies getting deleted too quickly, we assure that each sub-policy gets a minimum amount of samples in the sampling process. We also bound the minimum variance of our Gaussian sub-policies to small values in order to avoid singularities. After the sampling process, we evaluate the quality of the exploration-free policy (i.e., without variance) found so far.

4.2 Feature Based State Representation

We use feature transformations for representing the value function as well as achieving a higher flexibility in the gating policy. Different feature representations can be used for the gating and the value function. In the presented experiments, the gating network $\pi(o|\boldsymbol{s})$ is constructed on squared expansion of the state space. Thus, the squared features are constructed as the vector of all linear and squared combinations of the state dimensions. For example, for a two-dimensional state space the feature vector would be given by

$$\boldsymbol{\phi}([s_1, s_2]) = [1, s_1, s_2, s_1^2, s_2^2, s_1 s_2].$$

While the squared expansion of the state space can also be used as feature representation for the value function, more complex tasks often benefit from a more flexible representation, such as a kernel based feature transformation. In our experiments we used either kernel based features or features based on a Fourier transformation as shown by (Konidaris et al., 2011). The kernel based features used a squared exponential kernel, i.e.,

$$[\boldsymbol{\phi}_V]_i(\mathbf{s}) = \exp\left(-\frac{1}{2}(\mathbf{s} - \mathbf{s}_i)^T \boldsymbol{\Lambda}^{-1}(\mathbf{s} - \mathbf{s}_i)\right),$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix denoting the bandwidth of the kernel and the index i iterates over the reference set of observed states.

Choosing sufficiently expressive features for the representation of the value function is crucial to the success of the learning algorithm. As an example, we can consider a contextual bandit task, where the robot has to select the number of steps to take to walk to a goal position. The context in this task is given by the robot's initial distance to the goal. If the robot has access to insufficient features, e.g., only a constant feature, it cannot differentiate between the different starting positions, and, hence, cannot learn to adapt its policy to the initial position. This effect is compounded in the infinite horizon setting, were successful policies often depend on reaching intermediate states with potentially low rewards on the path to the goal states.

4.3 Model Learning

For the finite horizon, as well as the for the infinite horizon formulation, HiREPS requires a transition model $\mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}}$. In this paper, we use a sample based transition model, which reproduces previously observed state transitions and does not necessarily generalize. Using a sample based transition model effectively results in computing $V(\boldsymbol{s}')$ directly based on one single observed state transition instead of computing the expectation $\mathbb{E}[V(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{a}]$ in the update, Equation (25), as well as in the corresponding dual function in Eq. (51) as given in Appendix A.3. Replacing the expectation in such a manner is only feasible if the controlled system has limited stochasticity. However, it is indeed still sufficiently robust to solve the real robot tasks as presented in the experimental Section. For systems with more stochasticity, van Hoof et al. (2015) show how to learn a better model that is able to generalize if sufficient data is available.

4.4 Sample Efficiency

One of the important trade-offs when using iterative policy update methods is the number of rollouts performed per iteration. More rollouts per iteration result in more stable policy updates but also increase the number of overall rollouts required to learn a task. When using sample based policy update techniques in high dimensional action spaces, the number of samples is crucial. Using fewer samples than action dimensions will lead to an underestimate of the variance or the variance might even collapse, as the new policy is just based on the available samples. In HiREPS, we are not restricted to using only samples from the previous iteration of rollouts when computing the sample weights. By considering samples from M multiple past iterations, we can stabilize the policy update while retaining a fast learning speed. As a general rule of thumb, keeping three times as many samples as used per rollout can stabilize the algorithm against aggressive choices in the number of rollouts per iteration (lower than number of parameters) or high ϵ (≥ 1.5). When keeping samples from previous iterations, we usually define our current state-action distribution $q(\boldsymbol{s}, \boldsymbol{a})$ to be given by the collection of these samples. Alternatively, importance weighting schemes can be employed to incorporate samples from previous iterations. However, in our experience, these importance weighting schemes are often prone to destabilizing the learning algorithm.

4.5 Parametrized Trajectories

For the results of this paper, we often rely on movement primitives that inherently encode temporal correlations, such that each sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ describes the parametrization of one movement primitives. We are using the extended version of DMPs by Kober et al. (2010a), that parametrizes trajectories using a combination of weighted basis functions as well as the end point

g and final velocity $\dot{g}$ of the trajectory. DMPs are based on a simulated spring damper system $\ddot{x} = \alpha\,(\dot{g} - \dot{x}) + \beta\,(g - x)$ with damping factor α and spring stiffness β. The spring damper system will result in a trajectory that reaches the desired goal and goal velocity in a direct manner. However, the spring damper system alone is insufficient to encode arbitrary shapes along the trajectory. To represent arbitrary shapes, DMPs modulate the dynamical system with a forcing function $f(z, \boldsymbol{v}) = \boldsymbol{\Phi}(z)^T \boldsymbol{v}$ that depends on the phase z of the movement, the basis functions $\boldsymbol{\Phi}(z)$ and the basis functions weights $\boldsymbol{v}$. Using the forcing function to deviate from the direct path of the spring damper system allows DMPs to generate complicated trajectory shapes. To ensure that the desired final goal position and velocity constraints are met, the forcing function depends on the exponentially decaying phase variable z, such that the forcing function does not influence the system towards the end of the trajectory.

If a demonstration trajectory, for example from kinesthetic teach-in, exists, basis function weights $\boldsymbol{v}$ that reproduce the shape of that motion as well as the goal position and velocity can be computed to initialize the learning process. After initialization, the basis function weights as well as the desired goal position and velocity are the parameters that the reinforcement learner optimizes over. In the presented robot learning experiments we learn one DMP per degree of freedom. It is important to note that the output of the DMPs only represents the desired trajectories, which are then usually tracked by an internal controller of the robot itself. This distinction is important because even if the robot is unable to track trajectories, e.g., due to gain or torque limitations, the RL agent can adapt the DMP such that the resulting trajectory still solves the task. When using DMPs, each sub-policy lasts until the DMP has finished execution.

4.6 Computational Complexity

The presented algorithm depends on the three key computations: solving the dual function for HiREPS, fitting a gating policy and fitting the individual sub-policies. The optimization of the dual problem can be performed using existing optimizers such as, for example, the Broyden-Fletcher-Goldfarb-Shannon algorithm. The optimization can often be accelerated by providing the first and second derivative of the dual function. While the actual complexity depends on which algorithm is used, we can analyze the problem itself. The number of open parameters in this optimization problem stays constant with the number of options. However, the number of open parameters depends directly on the dimensionality of the features $\boldsymbol{\phi}_V(\boldsymbol{s})$ for the value function. Thus, the complexity of the optimization problem depends on the feature representation rather than on the number of options.

Fitting a gating policy based on the weights computed through the HiREPS optimization problem yields a multiclass classification problem. For example, multiclass logistic regression can be used. As before, a solution can be found through the use of third party optimization software. However, the number of open parameters scales multiplicative with the dimensionality of the gating feature representation and the number of options. Thus, for high dimensional gating features and a large number of options, the gating optimization problem can become computationally expensive.

Finally, the sub-policies have to be fitted. Given the gating, the sub-policies are independent and each sub-policy can be fitted individually using a weighted maximum-likelihood estimate. In the case of linear Gaussians, the complexity of this operation is governed by a linear regression operation which depends on the dimensionality of the sub-policy feature representation. However, since these optimizations are performed individually per option, the computational effort for this last step is usually negligible.

4.7 Hyperparameter Tuning

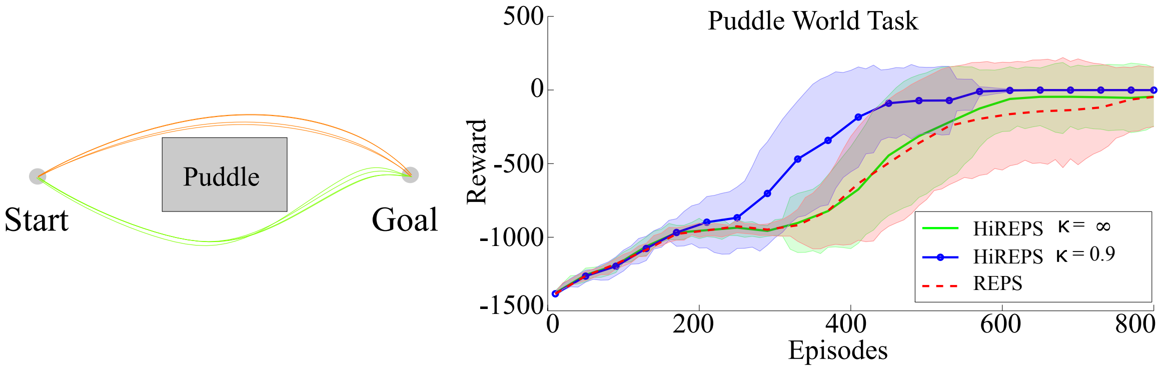
The presented method exposes three main hyperparameters to be set by the practitioner. These are the number of options O, the entropy bound κ as well as the relative entropy bound ϵ. The number of options can usually be chosen generously, i.e., around 20 seems to be reasonable for a wide range of problems. The algorithm will prioritize more promising options such that even if too many options are initialized, only options which yield high rewards will be sampled from after the first few iterations. The entropy bound κ is probably the most important parameter to consider since it does not have a clear equivalent in existing approaches. However, our experiments showed that a value of 0.9 seems to work well in almost all cases and no major tuning was necessary. The parameter ϵ is probably the parameter that is the most tuning-intensive in the proposed method, especially if the total number of episodes is crucial, e.g., in real robot experiments. In our experience values for ϵ between 0.5 and 1.5 are reasonable and, most often, we would start a new task with $\epsilon = 1$. Changing these parameters certainly influences the learning speed of the proposed method. However, while sub-optimal settings may lead to slower convergence, they usually do not prevent successful learning. Thus, in our experience, the algorithm is generally robust in that even sub-optimal settings will lead to convergence.

5. Experimental Validation

We validated the different presented formulations of HiREPS on both simulated and real robot tasks. We compare our algorithm to the non-hierarchical counterpart, REPS (Peters et al., 2010). Evaluating against REPS is interesting, as the two algorithms share the same basis. In effect, REPS is equivalent to our algorithm with just one sub-policy. This similarity allows us to directly investigate the influence of adding a hierarchical policy representation without additional confounders. The experimental section is structured analogously to the structure of Section 3, i.e., we first report results on the contextual episodic settings, subsequently the skill sequencing problem and finally the infinite horizon setting. For the contextual setting as well as for the skill sequencing, we report real robot results. In both cases, we start by presenting results on related simulated experiments to better investigate the properties of the presented algorithms as well as to compare to the REPS algorithm. For all presented results, we have optimized the parameters of the algorithms to deliver the best performance and both algorithms receive the same number of samples per iteration. For all experiments, if not noted otherwise, the experiments were repeated ten times to produce the errorbars reported in the results.

5.1 Episodic Skill Based Tasks

We start by presenting results for the episodic formulation of HiREPS. In the contextual setting, as well as for the skill sequencing setting in the next part, we start by presenting results of simpler experiments that allow a more in-depth analysis of the algorithms and then proceed to more complex tasks.



(a) Puddle World

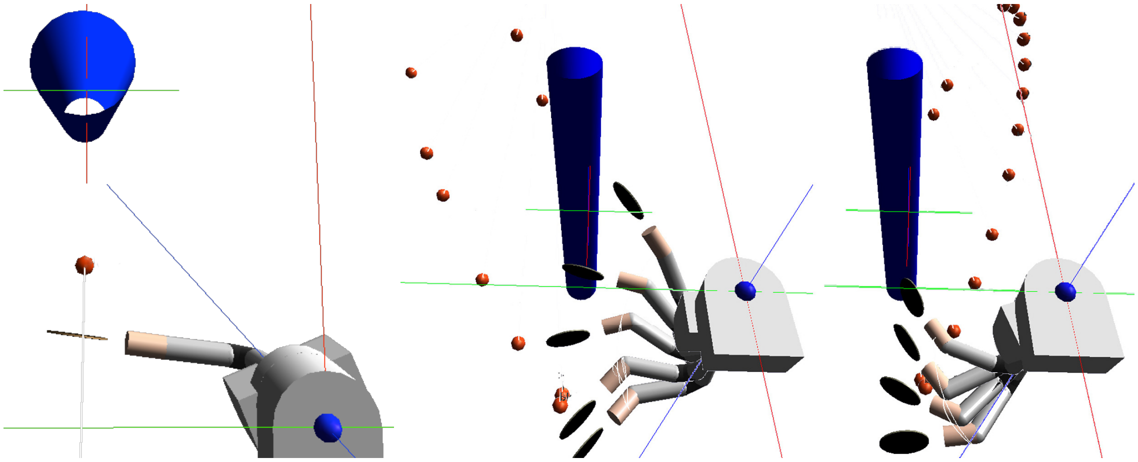
(b) Evaluation

Figure 3: (a) The puddle world task. Sub-policies are represented as two-dimensional DMPs with fixed end points. The DMPs have five basis functions per dimension and we learn the weights of the basis functions for the y dimension while leaving the weights for the x dimension fixed. The plot shows trajectories sampled from two sub-policies found after 80 iterations (using 10 samples per iteration) of HiREPS. Start and end points of the trajectory are pre-defined, the agent may choose the path in between those points. The puddle world has two solutions. The plot shows samples from both modes acquired by HiREPS after 30 iterations. (b) Performance of HiREPS and REPS on the puddle world task. As HiREPS can represent both solutions, it does not get stuck averaging over both modes. Note, that this effect is much more pronounced if we bound the entropy of the sub-policies.

5.1.1 PUDDLE WORLD EXPERIMENT

In a first toy task, we test HiREPS on a variation of the puddle world (Sutton, 1996). While this task is of limited difficulty it is interesting as it is a well known setting which exhibits the averaging problem of interest to us. Additionally, the simplicity of the problem allows us to thoroughly assess the quality of the solutions found by the RL agent, which is often difficult in real robot tasks. Our version differs from the standard version by having a continuous action space instead of a discrete one. While the agent proceeds with a constant velocity along the x-dimension of the environment, it has to learn the DMPs shape parameters such that the puddle is avoided. Thus, the actions $\mathbf{a}$ of our sub-policies are five-dimensional vectors. The reward of the task is given by the negative length of the line segments, which encourages shorter solutions. An additional punishment occurs for passing through the puddles. The arrangement of the puddles can be seen in Figure 3a. The presented puddle world has two solutions which are located close to each other. Still, the mean of both solutions leads through a puddle and, therefore, yields lower rewards.

In Figure 3b, we evaluate the performance of REPS and HiREPS with and without bounding the sub-policies' entropy. For HiREPS, just two sub-policies were used. REPS takes a longer time to reliably find good solutions, as the algorithm averages over both modes. HiREPS without bounded entropy performs slightly better than REPS. However, the advantage of HiREPS is much more pronounced when also bounding the expected entropy of the responsibilities. Furthermore, if we limit the entropy, the algorithm is able to reliably find both modes. The results also show that



(a) Top View

(b) Left and right swings

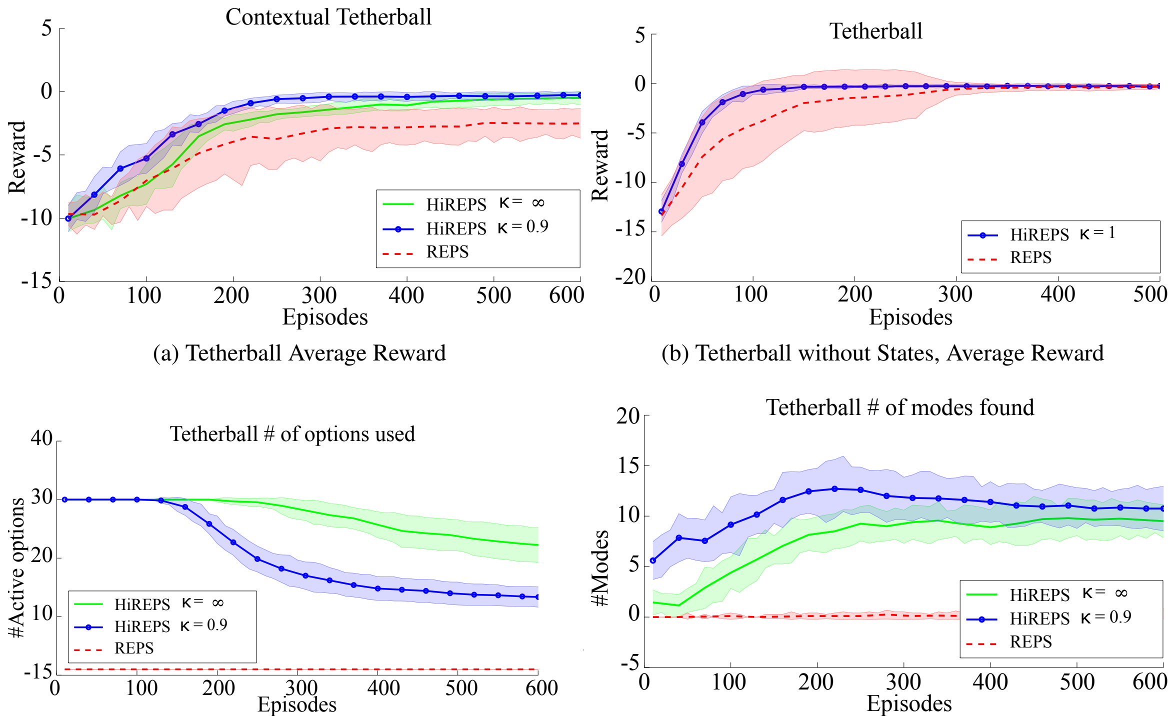
Figure 4: (a) Top view of the setup of robot tetherball. The ball is hung on a string from the ceiling in front of the pole. (b) Time series overlay of two swings in simulation, one left swing and one right swing. The two solutions use different sub-policies of the same hierarchical policy. HiREPS can also keep more sub-policies representing additional solutions to a task.

even after 80 iterations (using ten samples per iteration), the REPS learning curve still exhibits some variance, showing that the algorithm has not fully converged.

5.1.2 TETHERBALL

Our aim is to adapt the game 'Tetherball' for a robotic player, as shown in Figure 4b. Specifically, the task consists of a ball, a rope, an obstacle (i.e., a pole) and a target. The ball is hung in front of the pole, in a line with the robot, the pole and the target. This setup requires the robot to induce a circular trajectory to be able to wind the ball around the pole. In the planar simulation, the robot can induce this angular velocity through the elasticity of the rope. In the physically accurate simulation as well as on the real robot, a pre-strike is necessary to achieve similar results. This task presents a versatile solution space, as many different strategies successfully hit the target. Our goal is to model this versatile solution space with HiREPS. In order to thoroughly assess the proposed method, we split our investigation of the tetherball task into three parts of ascending difficulty. In a first evaluation, we implement a planar simulation of the task in which we abstract the robot into a force, i.e., we do not simulate a robotic arm but instead allow the agent to directly exert a force onto the ball. We use this simplified setup to compare HiREPS to its non-hierarchical counterpart as well as to investigate the effect of different settings. In a second evaluation, we use a detailed physical simulation of the task including the robot arm, in which we evaluate the benefits of a hierarchical policy in a more realistic setting. Finally, we present the results of the actual robotic task which show that HiREPS finds multiple solutions within one learning scenario.

Planar Simulation. For the purpose of testing the HiREPS method, we use a strongly simplified setup where the tetherball task is implemented in a planar simulation. We initialize HiREPS with 30 randomly located sub-policies and use ten samples per iteration. The agent can accelerate the ball



(a) Tetherball Average Reward

(b) Tetherball without States, Average Reward

(c) Tetherball Number of Sub-Policies Used

(d) Tetherball Number of Modes Found

Figure 5: (a) Average reward gathered by the REPS and the HiREPS with and without bounding the entropy of the sub-policies in the tetherball task including states. As REPS only uses a single sub-policy and thus a linear model as policy, it cannot represent the complicated structure of the solution. The HiREPS approach benefits from bounding the entropy of the sub-policies. (b) Average reward in the tetherball task without states. While REPS also finds one of the optimal solutions, HiREPS benefits from its structured policy representation and outperforms REPS in learning speed. At the same time, HiREPS finds both solutions. (c) Number of sub-policies used by the HiREPS approach with and without bounding the entropy. If the prior $p(o)$ of a sub-policy becomes too small it gets deleted. By bounding the overlap of the sub-policies, less sub-policies are used while the performance of the algorithm is increased. (d) Number of modes found by HiREPS with and without bounding the overlap of the sub-policies. We can see that, despite that HiREPS with bounding the overlap uses less sub-policies, it can find more modes. Without bounding the overlap of the sub-policies, many sub-policies concentrate on the same mode, which attenuates the advantages of the structured policy representation.

with a two dimensional impulse $\{F_x, F_y\}$. The reward is given by the negative minimum squared distance of the ball to the target throughout the ball's trajectory. The initial state of the agent is given by the initial position of the ball before hitting it. We only vary the x-position of the ball and learn different solutions to hit the target. As before, we compare HiREPS with and without bounding the overlap of the sub-policies to the standard REPS approach. In Figure 5a, we evaluate the average reward of all three approaches. The results show that HiREPS with the bound on the overlap outperforms the two other methods. The HiREPS approach already without the additional bound is better than the REPS approach, as REPS only uses one sub-policy to cover the whole

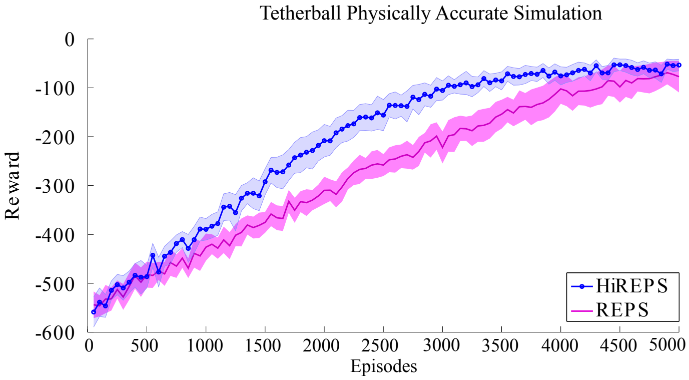


Figure 6: Results of the physically accurate tetherball simulation. Both REPS and HiREPS reach the same asymptotic reward. However, since HiREPS can differentiate between the multiple local optima and assign different options to these optima, it exhibits faster learning speed.

state space. Thus, REPS needs to approximate the optimal policy using a single linear model. Therefore, in order to have a fair comparison to REPS, we also compare HiREPS and REPS on the tetherball task without states, i.e., we always start from the same initial state in the middle. This comparison can be found in Figure 5b. We can observe that REPS is impaired by the multi-modality of the solution space as it tries to average over several modes, but finally learns a solution that is equally good as the solutions found by HiREPS. However, while REPS only learns one solutions, Figure 5d shows that HiREPS learns multiple solutions at the same time. In order to compute the number of modes, we divide each dimension of the state action space into 5 partitions and count the number of partitions which contain at least one sub-policy with an average reward larger than -1. The plot shows that, as the sub-policies distribute more uniformly in the state-action space due to the bounding, we can find more modes. Thus, the bound of the overlap also helps us to find more versatile solutions as it avoids situations where multiple sub-policies concentrate on the same solution. In Figure 5c, we show the number of sub-policies used for different bounds of the overlap $\tilde{\kappa}$. By bounding the overlap, the gating network can learn to select individual sub-policies more decisively, explaining the faster learning speed in the experiments when using the bound.

Physically Accurate Simulation. Having compared the properties of (Hi)REPS on the simpler simulation, we proceed to presenting the results of the physically accurate simulation as shown in Figure 4a, which also serves as a stepping stone to the real robot results. As described, the pole is placed on a line between the ball's resting position and the target location, such that it is impossible to hit the target with a single strike of the ball in direction of the target. In order to evoke a circular trajectory that arcs the ball around the ball, displacing the ball from its resting pose is necessary. Thus, we decompose our movement into a swing-in motion and a hitting motion. However, the parameters for both motions are combined into one parameter vector that is learned jointly. A more powerful approach would be to respect the natural separation of the successful trajectories and use the skill sequencing approach as presented in Section 3.2. The reward is determined by the speed of the ball when the ball winds around the pole. We define winding around the pole as the ball passing the pole on the opposite side of the pole. We run the algorithms with 50 samples per iteration and always keep the last 400 samples. We initialize our algorithm with 30 sub-policies and stop deleting sub-policies if only 5 sub-policies are left. The resulting learning curve in the simulation can is shown in Figure 6. After 100 iterations the robot has learned to wind the ball around the pole in 5/5 trials. In all trials, we were able to observe sub-policies for the left and for the right mode. The resulting movements are shown in Figure 4b and illustrate that the resulting movement of the two solutions are easily differentiated. The results show, that albeit HiREPS uses the same amount of

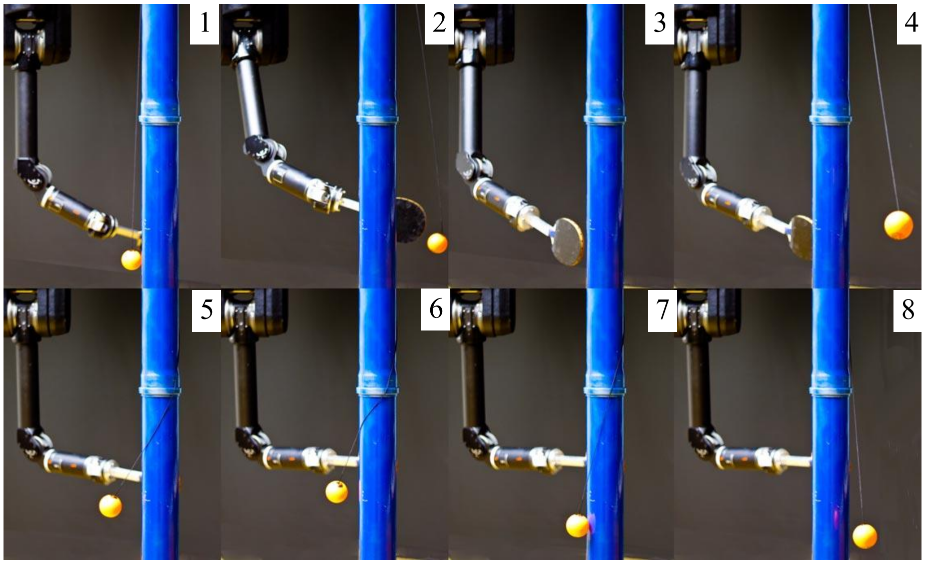


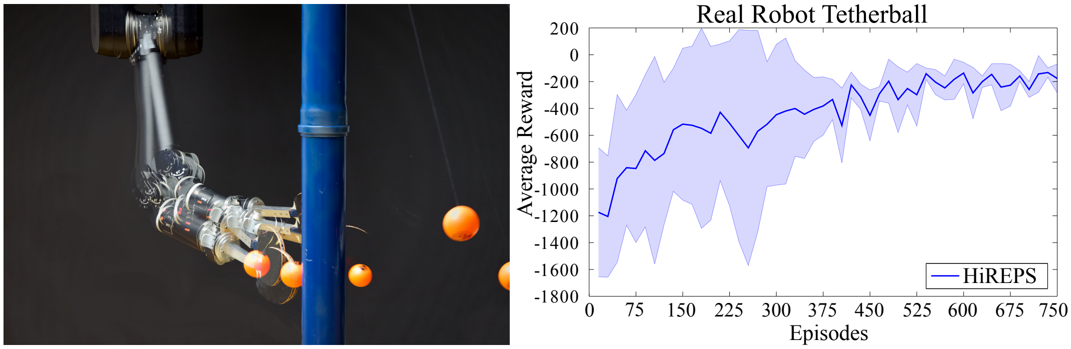
Figure 7: Time series of a successful swing of the robot. The robot first has to swing the ball to the pole and can, once the ball has swung backwards, arc the ball around the pole.

total samples per iteration as REPS, it can use those samples to learn multiple solutions while being faster than REPS can learn a single solution.

Real Robot Tetherball. After presenting the results on the two simulated tetherball settings, we proceed to show the results of the real robot experiment. For the robot experiment, we mounted a table-tennis paddle to the end-effector of the robot arm. The real-robot setup is depicted in Figure 8a and two successful hitting movements of the real robot are shown in Figure 7. In order to track the ball, a Kinect RGBD camera was setup to look at the robot from the opposite side of the pole. The vision data was used to compute the reward signal. As in the physically accurate simulation, the robot needed to perform a pre-swing as well as the actual swing. However, the real robot can easily be bootstrapped through imitation learning. Thus, we extract the shape parameters $\boldsymbol{v}$ by kinesthetic teach-in (Ijspeert and Schaal, 2003) for both motions. Subsequently, the robot learns the final positions and velocities of all seven joints through the presented approach. Additionally, we learn the waiting time between both movements. This task setup results in a $2 \times 2 \times 7 + 1 = 29$-dimensional action space. We initialized the algorithm with 15 sub-policies and sampled 15 trajectories per iteration. While this scheme amounts to considerably fewer samples than in simulation, it is sufficient to learn multiple solutions, given the initial demonstrations. We performed three trials to produce the errorbars reported in the results. The learning curve is shown in Figure 8b. The noisy reward signal is mostly due to the vision system and partly also due to real world effects such as friction which lead to a non-repeatability of rollouts. Two resulting movements of the robot are shown in Figures 7 and 4b.

5.2 Planar Reaching Task

To further evaluate the properties of the proposed algorithm, we present a series of experiments on a planar reaching task. In this simulated task, the agent controls the joint trajectories of a three-link



(a) Tetherball

(b) Real robot results

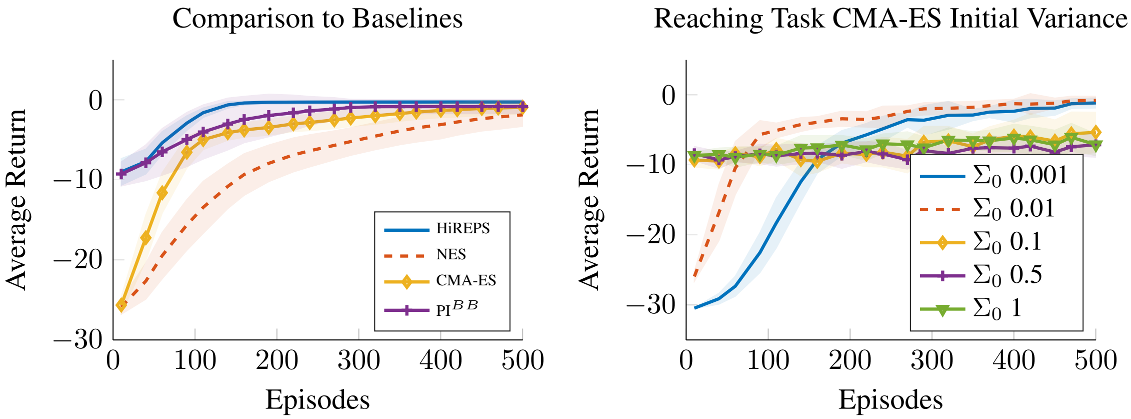
Figure 8: (a) The real robot tetherball setup. (b) Average rewards of HiREPS on the real robot Tetherball setup. Mean and standard deviation of three trials are shown. In all three trials, the robot has found solutions to wind the ball around the pole on either side after 50 iterations.

robotic arm using DMPs with two basis functions per joint. The robot starts from a fixed initial position and executes a trajectory for 100 time steps. At time steps 40 and 100 the robot is rewarded for passing through via points. While the robot is controlled in joint space, these via points are given in task space. Furthermore, for both time steps two possible via points exist.

Comparison to Baseline Methods. Most of the presented experiments compare the performance of HiREPS to its natural competitor, REPS. To better analyze the performance levels of HiREPS, we also performed an experiment comparing HiREPS to other state of the art methods. Specifically, we compared to the PI^{BB} algorithm (Stulp and Sigaud, 2013), the CMA-ES algorithm (Hansen et al., 2003) as well as the NES algorithm (Wierstra et al., 2014). To compare HiREPS against these baselines we performed an empirical optimization of the open parameters these methods perform. For example, Fig. 9b shows the effects of the initial variance of CMA-ES. The results in Fig. 9a show that HiREPS converges faster than the alternative algorithms we compared to. Furthermore, the alternative algorithms are designed to represent a single solution. In this experiment, all algorithms used 10 samples per iteration. Both HiREPS and PI^{BB} used a higher initial variance than CMA-ES and NES in this experiment since the latter methods' performance decreases using higher initial variances as shown in Fig. 9b. Due to this higher initial variance, PI^{BB} and HiREPS start with a higher initial reward.

Initialization of the Algorithm. Since the presented algorithm is based on the availability of multiple sub-policies, these options have to be initialized to sensible values. In the presented experiments linear Gaussians were usually used as sub-policies, i.e., $\pi(\boldsymbol{a}|\boldsymbol{s}, o) = \mathcal{N}(\boldsymbol{a}|\boldsymbol{\mu}_o + \boldsymbol{F}_o\boldsymbol{s}, \Sigma_o)$. To initialize each linear Gaussian, we can keep $\boldsymbol{F}_o = \boldsymbol{0}$ and $\Sigma_o = c\boldsymbol{I}$, where the constant c is a task specific variable set by the experimenter. While F_o and Σ_o are initially identical for all sub-policies, the means $\boldsymbol{\mu}_o$ have to be different to allow for later separation of the sub-policies based on the responsibilities. To that effect, we usually sampled the individual sub-policy means $\boldsymbol{\mu}_o$ from a normal distribution, i.e., $\boldsymbol{\mu}_o \sim \mathcal{N}(\cdot|\boldsymbol{0}, \Sigma_\mu)$.

Figure 10a shows the results of varying Σ_μ relative to the admissible range of parameters. The results show that while a larger initial separation seems to help bootstrapping the learning process,



(a) Comparison To Baselines

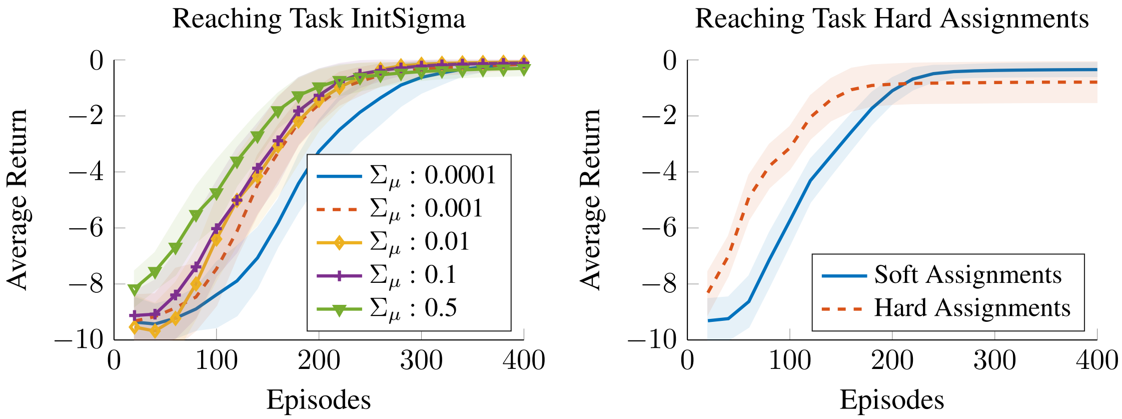
(b) Influence of Initial Variance in CMA-ES

Figure 9: (a) Comparison of HiREPS to popular baselines. HiREPS and PI^{BB} start at higher initial rewards because CMA-ES as well as NES have to be initiated with a much lower variance. While HiREPS and Pi^{BB} start with a high-variance initial policy (or sub-policies) and keep contracting it until convergence. CMA-ES and NES on the other hand work best with a smaller initial variance that is kept for longer. (b) Effects of different initial variances on CMA-ES. We chose the best initial variance for the comparative experiments.

the overall effect of changing this parameter is negligible. In this experiment, a total of 20 options were available to the algorithm. While the comparison to the base-line methods was performed using ten samples per iteration, for the following experiments 20 samples per iteration were used. Using more samples per iteration allowed us to observe the effects under investigation more clearly, while the comparison to other baselines was optimized for the CMA-ES and NES algorithms.

The Influence of Probabilistic Option Assignments. The presented algorithm allows for inter-option learning, i.e., the sharing of information between options during learning which is expected to improve learning. To test this hypothesis, we performed an experiment comparing the proposed formulation of the algorithm to an alternative formulation based on hard assignments. In the alternative formulation, every option was effectively limited to using it's own samples to update its policy. This behavior was achieved by assigning a responsibility of 1 to the option that generated a sample while all other options had zero responsibility for the same sample. Using soft assignments allows the gating to shift responsibilities such that options can specialize on distinct sub-tasks. The results in Figure 10b show that using soft assignments did indeed improve asymptotic learning performance but was slower in the early stages. However, in our experiments we observed that the importance of this effect would diminish with an increasing number of options. As the number of options increases, HiREPS can implement a more effective divide and conquer strategy. Sharing of information can becomes less important after the division of the state-action space has been successfully performed.

Evaluation of the Entropy Bound. One of the main parameters to be considered in the proposed algorithm is the desired bound on the entropy of the responsibilities, $\tilde{\kappa}$. The lower the value for $\tilde{\kappa}$, the more aggressively the algorithm will separate the individual options. Figure 11a shows the



(a) Effect of Different Initializations

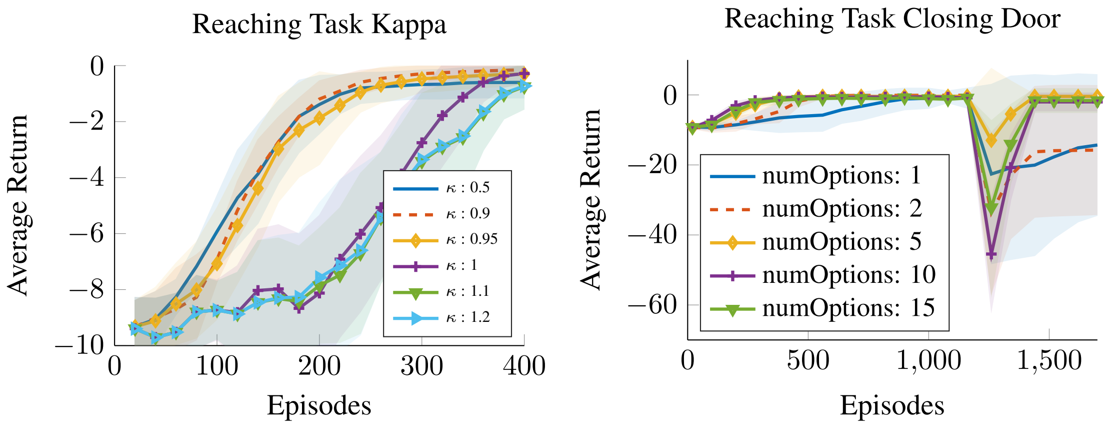
(b) Effect of Probabilistic Responsibilities

Figure 10: (a) Investigation of different initial distributions of sub-policies. Using a low initial variance of the distribution over option means, all options share similar responsibilities for all samples and the entropy bound forces a separation of the options. Using a higher initial variance for this distribution, the individual sub-policies will be responsible for different regions of the action space from the beginning. While the results show that such an initial separation can improve learning speed initially, it can also lead to sub-optimal asymptotic performance. (b) Using hard assignments, i.e., not sharing information between options, leads to high initial learning speed. However, using the hard assignments, the algorithm did not always find the optimal solution. In our experiments, the non-probabilistic (hard assignments) version of the algorithm would usually require more samples per iteration to consistently find equally good solutions as the proposed probabilistic approach. However, our experience shows that when using more samples per iteration, the learning speed of the hard assignment approach will actually decrease relative to the speed of the probabilistic approach. This effect can be explained by the fact that each option is drawn to all local optima more uniformly if more samples are used.

results of evaluating a wide range of possible parameter values for $\tilde{\kappa}$ in the reaching task. The results show that values for $\tilde{\kappa} \geq 1$ yield slower convergence speeds while the options still overlap and aim to explain multiple solutions. As in most other experiments presented in this paper, a value of $\tilde{\kappa} = 0.9$ seems to yield the best learning speeds. While even lower values of $\tilde{\kappa}$ do not noticeably decrease the learning speed in the presented experiment, our experience shows that lower values may prevent successfully learning multiple solutions. Since lower values of $\tilde{\kappa}$ force options apart, only few options may actually be fully developed while the probability of sampling from most other options will diminish.

The results reported for this experiment and the puddle world experiment regarding the entropy bound mirror our experience with the remaining experiments reported in this paper. In all experiments a bound of $\kappa = 0.9$ seems to give good results and much higher (> 0.95) or much lower (< 0.8) values usually deteriorate performance. Thus, for the remaining experiments, we chose $\kappa = 0.9$ and do not present further evaluations.

Robustness to Changing Environments. One main contribution of the proposed algorithm is to learn multiple solutions for the same task. Learning multiple solutions can be interesting if, for



(a) Influence of Entropy Bound

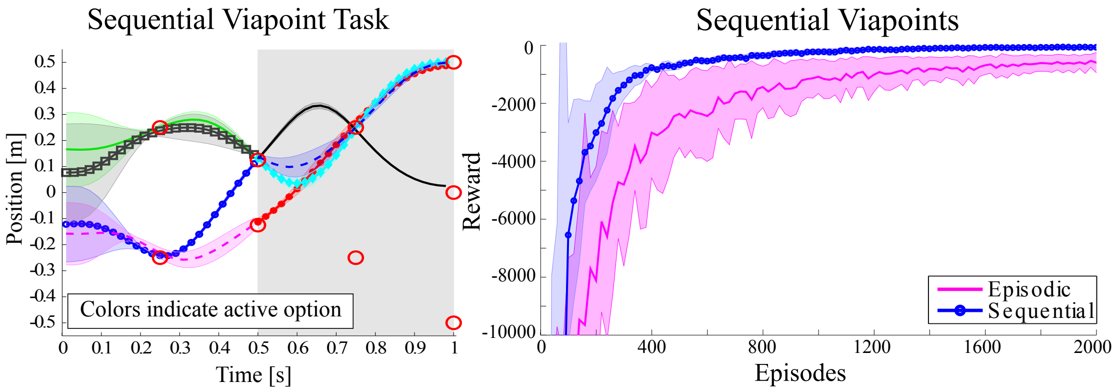
(b) Changing Environments

Figure 11: (a) Comparison of different relative values κ for the entropy bound. The results show that higher values for the entropy bound slow down learning, as the options are not forced to separate. Thus, multiple options will compete for the same local optimum for longer. Very low values for κ will also slow down learning and can potentially lead to sub-optimal asymptotic performance when the entropy bound ‘overpowers’ the reward maximization criterion. (b) Investigation of a task which requires multi-modal solutions. After learning a solution, some of the via points were randomly disabled and the agent had to rely on the availability of alternative solutions. The results show that increasing the number of options allowed for a robust policy which could recover from the change in the environment because the agent had previously learned multiple solutions.

example, the environment changes in a way that makes some solutions inaccessible. To evaluate the behavior of the proposed algorithm in such a scenario, we let the algorithm learn a solution for the reaching task as described above. After 30 iterations HiREPS typically converged to a good solution. At that point, the via points on either the lower or upper path were randomly disabled, which simulated blocking one of the paths. Figure 11b show the effects of learning multiple solutions. With only one option, the agent cannot recover in about half the trials. Using more options increases the likelihood that the agent has learned sufficiently versatile solutions to successfully perform the task even after cutting off of the paths.

5.3 Sequencing of Skills

As evident from the tetherball experiment, many real world tasks require multiple steps to be solved. To evaluate the skill sequencing implementation of HiREPS presented in Section 3.2, we present a second set of experiments. Before testing skill sequencing on a real robot task, we evaluated the time-indexed HiREPS algorithm on a via-point task in order to illustrate the properties of the approach. In this task, we modeled a second-order dynamical system. The state of the agent is given by its position x and velocity $\dot{x}$. The actions u control the accelerations $\ddot{x}$. The task of the agent is to reach specified via-points at four different points in time. For each of these time points, different via-points exist. The reward at the time points is given by the negative squared distance to the closest via-point. In addition to the deviation to the via-points at the four specified time



(a) Task Setup. Colors indicate active option.

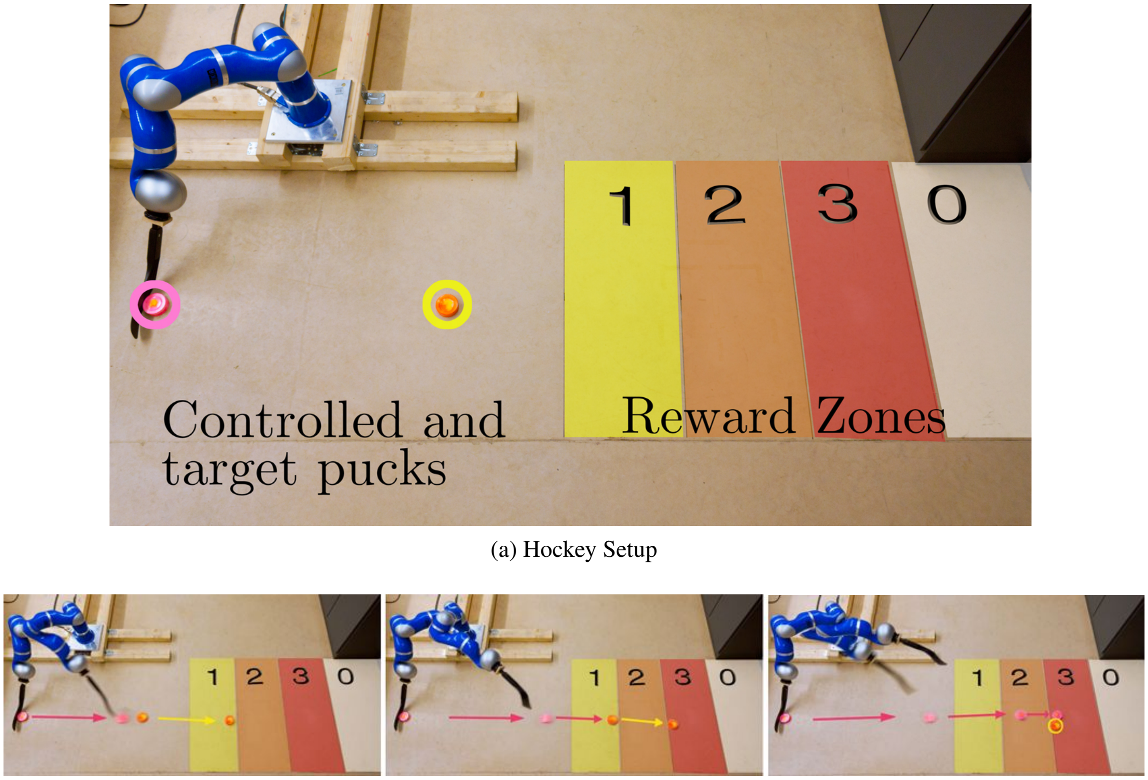
(b) Average Reward.

Figure 12: (a) We use this via-point task to illustrate our algorithm.The agent has to reach one of the via-points (denoted by red circles) at each of four specified times $[0.25, 0.5, 0.75, 1.0]$s. The reward is given by the negative squared distance to the closest via-point. The last via-point has to be reached with zero velocity. The initial positions and velocities are sampled from a Gaussian distribution with zero mean and a standard deviation of 0.25 for the position and 0.1 for the velocity. In this task, we learn to sequence two motor primitives, with the second primitive starting at $t = 0.5$s (shaded region in which the line colors change). This task is per construction multi-modal and illustrates how our algorithm learns distinct motor primitives. The mean and variance are indicated by shaded error bars. The agent learned several but not all possible solutions to solve the task. (b) The multi-modal via-point task learned with episodic and sequential motor primitive learning where the movement was decomposed into two primitives, see Figure 12a for a more detailed description. As we can see, our algorithm could exploit this decomposition resulting in increased learning speed and higher quality final policies.

points, the reward function contains a squared punishment term for taking high accelerations. As we defined multiple via-points for each of the four time-points, this task has multiple solutions per construction. The agent used 20 samples per iteration for this task. The exact setting of the task including its via-points is depicted in Figure 12a.

In order to demonstrate sequencing of skills, we decomposed the task into two DMPs which were executed sequentially. We used five shape parameters for both DMPs. In addition, we also learn the goal-parameter of the DMP, resulting in 6 parameters per movement primitive. To compare our sequencing method to the commonly used episodic policy search setup, we also solve this task with the episodic version of HiREPS. In this case, we only used one DMP with ten shape parameters and the additional goal-parameter. For both scenarios, the agent could choose between four distinct sub-policies o_i at each decision time-point. As we can see from Figure 12a, the agent learned to select these primitives according to the state at the decision time-points as well as to adapt the primitives such the task can be solved. Our approach was able to learn multiple solutions for the task as can be seen from Figure 12a. However, only a subset of all possible solutions was found.

The comparison of the episodic and the sequential learning methods can be seen in Figure 12b and shows the advantage of the sequencing approach in learning speed as well as in the quality of the learned solution. Additionally, the episodic formulation will also require more options in total



(a) Hockey Setup

(b) First strike

(c) Second strike

(d) Final strike

Figure 13: (a) The robot hockey task. The robot has two pucks, the pink control puck and the yellow target puck. The task is to shoot the yellow target puck into one of the colored reward zones. Since the best reward zone is too far away from the robot to be reached with only one shot, each episode consists of three strikes. After each strike the control puck is returned to the robot, but the target puck is only reset after one episode is concluded. Concluding an episode with the target puck in one of the reward zones yields rewards from one to three as indicated in the picture. However, if the robot shoots the target puck too far, the reward is zero. (b-d) One episode of the Hockey task, consisting of three strikes. Each picture shows the initial and final position of control and target puck. The movement of the pucks is indicated by arrows. The robot can shoot the pink control puck to move the target puck and tries to place the yellow target puck in one of the marked target zones while not overshooting. In the depicted episode the robot only needed two strikes to place the target puck into the highest reward zone. With the last strike the robot taps the target puck only slightly without actually moving it to avoid negative reward for missing it.

in order to find all possible solutions than the sequential formulation. However, in the presented experiment we only rewarded the agent for finding at least one solution such that this effect did not alter the resulting performance analysis.

5.3.1 EVALUATION ON THE ROBOT HOCKEY TASK

As before, after using a simpler task to investigate the properties of the proposed algorithm, we now present the results of a more sophisticated task. Similar to the structure of the tetherball experiments,

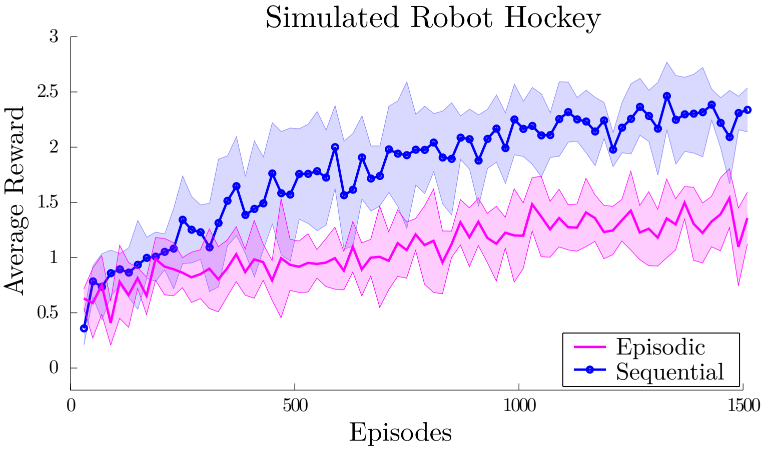


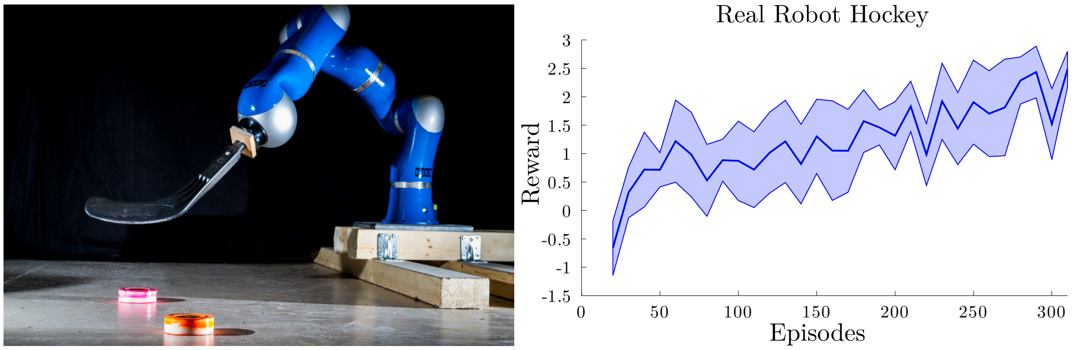
Figure 14: Comparison of sequential motor primitive learning to the episodic learning setup on the simulated robot hockey task. The sequential motor primitive learning framework was able to find a good strategy to place the puck in the third reward zone in most of the cases while the episodic learning scenario failed to learn such a strategy.

we first present a comparative analysis on a physically accurate simulation and then proceed to show the results of the real robot task. In the robot hockey task, the robot has to move a target puck into one of three target areas. This target puck can only be moved by shooting a control puck at it. The target areas are defined by a specified distance to the robot. The first zone is defined as distance from 1.4 to 1.8m, the second zone from 1.8 to 2.2m and the last zone from 2.2 to 2.6m. The robot gets rewards of 1, 2, and 3 for reaching zone 1, 2 or 3, respectively, with the target puck. If the robot overshoots the last target zone, the reward drops to zero. The reward is only given after each episode which consists of three shots of the control puck. After each shot, the control puck is returned to the robot. The target puck, however, is only reset after each episode. The setup of the robot hockey task is shown in Figure 13a.

The 2-dimensional position of the target puck defines the state of the environment as perceived by the agent. After performing one shot, the agent observes the new position of the target puck to plan the subsequent shot. In order to give the agent an incentive to shoot at the target puck, we punished the agent with the negative minimum distance of the control puck to the target puck after each shot. While this reward was given after every step, the zone reward was only given at the end of the episode (every third step) as $r(\boldsymbol{s}_{K+1})$.

We used a DLR-Kuka lightweight arm with 7 degrees of freedom as depicted in Figure 15a. We used DMPs to represent single motor primitives where we only adapted the goal positions and velocities of the primitives. This setup resulted in 14 parameters per primitive per shot. Thus, the episodic version of HiREPS has to optimize one 42-dimensional parameter vector while the time indexed version of HiREPS that we use for skill sequencing has to optimize three policies with 14-dimensional parameter spaces each. Both, the episodic as well as the time-indexed version of HiREPS had access to five options. The shape parameters $\boldsymbol{v}$ of the single primitives were learned from imitation by collecting trajectory data via kinesthetic teach-in.

Simulation Results. We first implemented a realistic simulation of the robot hockey task. In simulation, we varied the initial position of the puck by sampling the position from a normal distribution with standard deviation of 10cm. The agent used 30 samples per iteration. We compared our sequential motor primitive learning method with its episodic variant. For the episodic variant, we encoded the policy-parameters of all three shots into one policy, resulting in 42 parameters. The episodic variant cannot use state-feedback except for the information of the initial position of the puck. To make the comparison as fair as possible we did not use any noise in our simulation



(a) Hockey Setup

(b) Real Robot Hockey Rewards

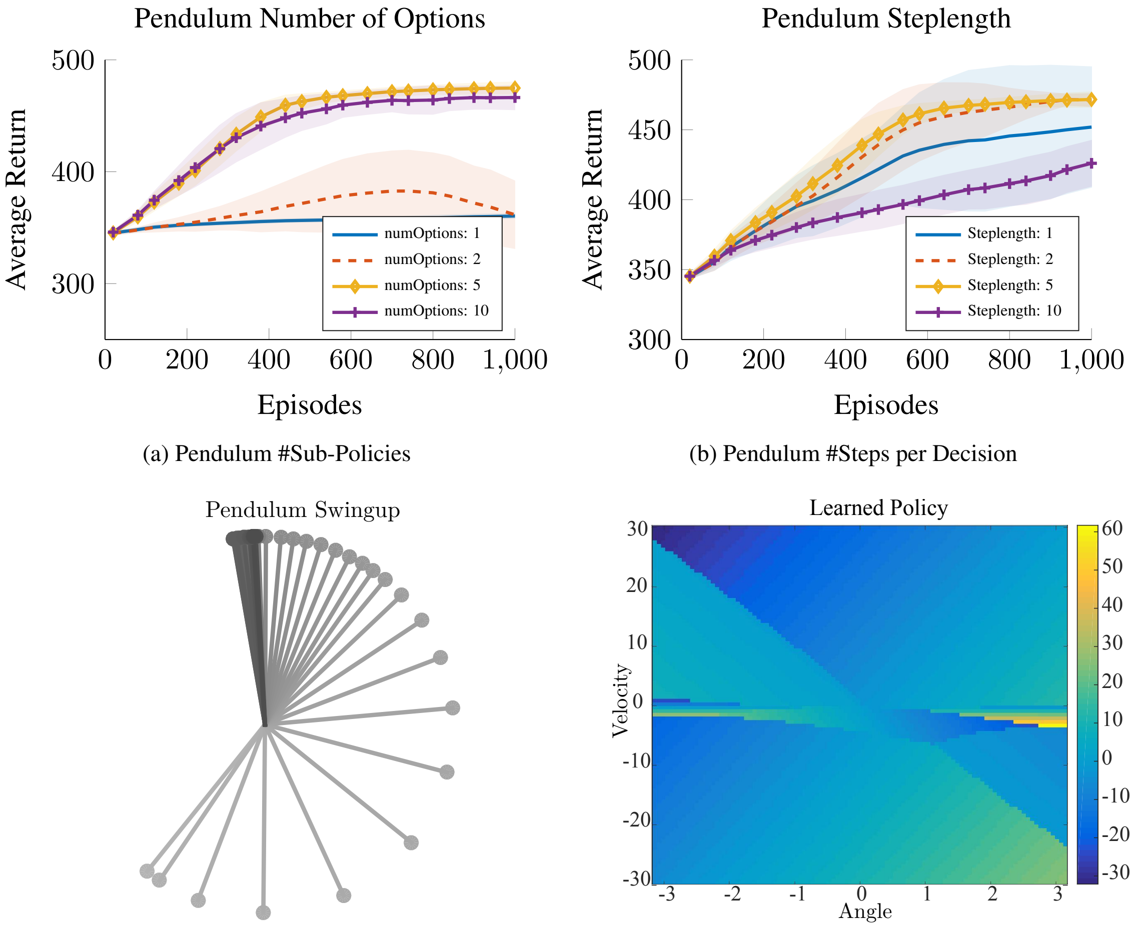
Figure 15: (a) The real robot Hockey setup. The robot first has to shoot the pink puck on the orange puck, to move the orange puck into a target zone. (b) One trial of of the real robot hockey tasks. The robot starts with a negative initial reward and learns to achieve an average reward of 2.5 after 300 episodes. The optimal theoretical reward of the presented task is 3.0. However, learning has not yet converged and had to be stopped prematurely due to time constraints.

and, hence, the initial position of the puck is sufficient to solve the task. The comparison of both methods can be seen in Figure 14. The episodic learning setup failed to learn a proper policy while our sequential motor primitive learning framework could steadily increase the average reward. Our method reached an average reward of 2.3 after learning for 1500 episodes. Note that an optimal strategy would have reached a reward value of 3, however, this is still a clear improvement in comparison to the episodic setup, which reached a final reward value of 1.4.

Real Robot Results We used a Kinect RGB-D camera to observe the state of target puck. For the real robot hockey task, the initial position was not varied. On the real robot, we could reproduce the simulation results. The robot learned a strategy which could move the target puck to the highest reward zone in most of the cases after 300 episodes, where the robot used ten samples per iteration. One episode of robot hockey is depicted in Figure 13. In the final trials, the robot tended to prefer using a soft hit in the first shot and to shoot the target puck to the last reward zone with the remaining two shots. This behavior yielded higher average reward, since it is easier to only tap the target puck without moving it too much while it is still closer to the robot. During learning the robot steadily adapted his strategy when it mastered the necessary motor skills to achieve higher rewards by placing the target puck in the highest reward zones.

5.4 Infinite Horizon Formulation for Sequencing Skills

We evaluated the infinite horizon formulation of HiREPS on the pendulum swing-up task in simulation. In this task, the pendulum starts hanging down with a random perturbation. The goal of the robot is to find a solution that first swings up the pendulum and then stabilizes the pendulum at the top. Instead of directly choosing motor commands in each time step, the robot chooses a desired joint value which is tracked with a PD-controller that is active over multiple time steps d (in the standard setting $d = 5$). While using direct motor commands is generally feasible as well, using a PD-control scheme is often beneficial from a robotics point of view. Real systems are often



(a) Pendulum #Sub-Policies

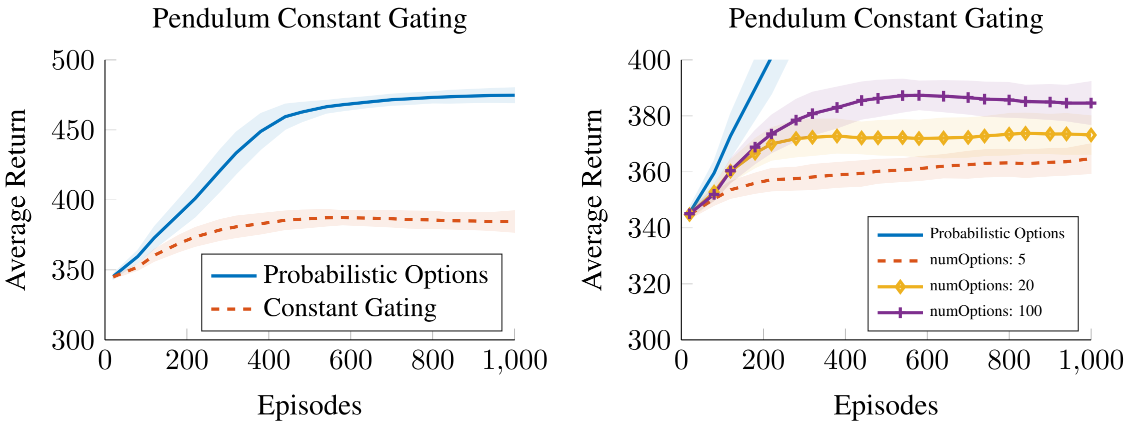
(b) Pendulum #Steps per Decision

(c) Task Setup

(d) Policy

Figure 16: Evaluations of the pendulum swingup task. The task starts with the pendulum hanging down and the robot has to swing up and stabilize the pendulum. (a) The results show that one linear sub-policy is insufficient to learn this task. With five sub-policies the robot is able to learn the task, with ten sub-policies the asymptotic performance does not improve anymore. (b) The number of time steps d each sub-policy stays active. The results show that activating a sub-policy for multiple time steps increases the speed of learning. If a sub-policy stays active for more than five time steps, the performance decreases. (c) One rollout of the pendulum swingup task. The pendulum starts at the bottom and requires a pre-swing to be brought to the upright position. (d) Pendulum Swingup-Policy learned by HiREPS. Colors indicate the mean value of the policy.

controlled at very high frequencies internally, however, policy signals are usually only necessary at lower frequencies. Reducing the control frequency of the policy will increase the signal to noise ratio, especially in real systems. Furthermore, even in simulated systems, the temporal extension of the actions can benefit the learning process. Since this task is highly non-linear in nature, it cannot be solved with a single linear policy. However, since HiREPS is able to represent a piecewise linear policy, it can be used to solve the pendulum swing-up task with multiple linear sub-policies. In this



(a) Comparison to Constant Gating

(b) Number of Options in Constant Gating

Figure 17: (a) Evaluation of a pre-defined constant gating vs. a learned gating. In the constant gating approach, the gating was pre-determined by performing a K-Means clustering of the state space. The results show that the pre-defined constant gating does not allow the agent to successfully learn the task. (b) Effect of the number of options when using a constant gating. The results show that the constant gating depends strongly on the number of options used, which related directly to the resolution with which the state-action space is parceled. Even with 100 options the constant gating approach is outperformed by the learned approach which uses only five options.

task, the pendulum has a mass of 10kg, a length of 0.5m and a friction coefficient of 0.2. The robot can exert at most 30nm of torque and uses 20 samples per learning iteration. The internal robot control runs at 100Hz and the restart probability in the base setting is given as $(1 - \gamma) = 0.02/d$, where d determines the control frequency of the learned policy. The primitive actions wich are chosen by the sub-policies are the desired joint angles of the pendulum. The robot can typically perform a swing-up within 1.5s. To represent the value function, we used the Fourier transform based feature transformation of the states using five Fourier bases, see Section 4. The gating uses a squared expansion of the states as feature representation. The reward function punishes deviation from the desired upright position with a factor of 500 and punishes velocities with a factor of 10. These punishments are subtracted from a base value of 500. The results of the experiments show that a single linear policy is insufficient for swinging up and stabilizing, however, HiREPS successfully combines multiple linear policies to solve the task, a resulting policy is shown in Fig 16.

Evaluation of a Constant Gating. In the pendulum swing-up task, a good gating policy is essential if the task is to be solved using simple sub-policies. To evaluate the effect of a learned gating versus a constant pre-determined gating, we compared the proposed approach to an alternative formulation where the individual action policies can be learned, but the gating stays constant. To find a such a constant gating, we divided the state space using K-Means clustering and kept the resulting gating constant afterwards. To perform K-Means clustering we sampled 1000 data points uniformly within $\pm[\pi, 50]$, where π in this case is the numerical value. The results show that the performance of such a constant gating strongly depended on the number of options available. However, even with 100 options the constant gating did not allow to learn this task successfully.

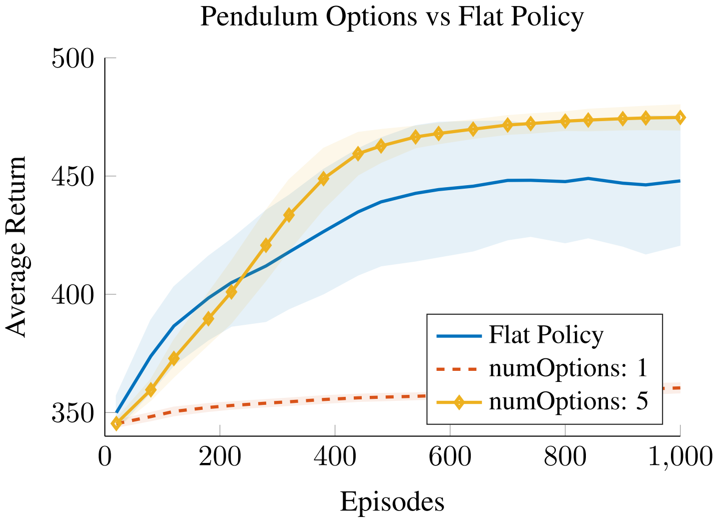
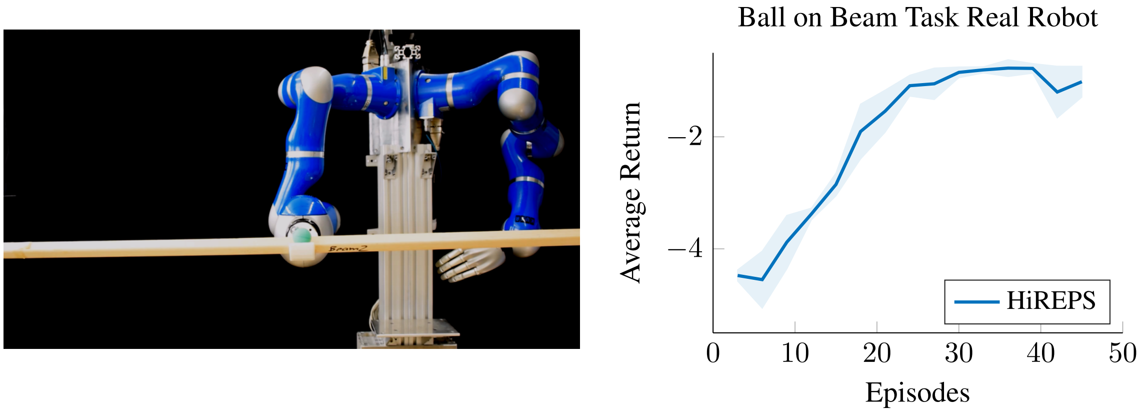


Figure 18: Comparison to flat but non-linear policy. The non-linear policy is given by a linear Gaussian based on a non-linear feature transformation of the states while the hierarchical policy uses only linear sub-policies. The results show that the non-linear flat policy is outperformed by the hierarchical policy. While the flat policy is able to learn the solution, the asymptotic performance varies over multiple runs and would require more samples per iteration to exhibit a robust learning performance.

Comparison to a Non-linear Flat Policy. While the proposed approach can learn the pendulum swing-up task using a combination of linear sub-policies, the task can also be solved using a single non-linear policy. To evaluate the effects of the proposed approach, we compared to a non-linear flat policy which was based on the same features as the gating in HiREPS. The results in Figure 18 show that while the non-linear flat policy initially learned faster than the HiREPS, the average asymptotic performance of the HiREPS was higher. Initially, HiREPS requires some iterations to learn a good gating policy. However, once a good gating policy was available, having different options that specialize on sub-tasks such as high accelerations vs. stabilization yielded better policies.

5.5 Infinite Horizon Real Robot Experiment

To evaluate the practical applicability of the proposed infinite horizon approach, we performed a real robot evaluation on the 'ball on a beam' task as shown in Fig. 19a. In this task, the robot has to guide a ball to a prescribed position on a beam and balance it at this position. In our experiments, the robot had to choose between two possible goal positions, 10cm to the left or right of the beam center. To learn the task, the robot learned a policy which set a new desired end effector angle every 0.5s. Every policy was evaluated for 20s and in each iteration the agent collected three episodes before updating the policy. In this task, the state space was given by the ball's position and velocity as well as the current beam angle. The sub-policies were linear Gaussian policies. The gating operated on squared features of the state space and the value function was based on a Fourier feature transformation of the state space using five basis functions. The reward function in this task was given as the summed negative squared distance and velocity of the ball (to the closest goal position) in every control step. The ball's position and velocity were determined using a Microsoft Kinect which was mounted above the robot. The results in Fig. 19b show that the robot was able to learn this task within 30 episodes in all three trials performed.



(a) Ball On Beam Setup

(b) Ball On Beam Results

Figure 19: (a) The experimental setup for the 'ball on a beam' task. A beam of 1m length is attached as the robot's end effector. The robot has to balance a ball in one of two goal positions, ±10cm from the center of the beam. The ball is initialized alternating on the right and left end of the beam. (b) Results of three trials on the real robot. In all trials the robot was able to learn the task within 30 episodes. In the second trial performance dropped slightly at the end of the trial before recovering again. This drop was most likely due to noise in the vision system.

6. Conclusion & Discussion

In this paper, we present a novel method, derived from first principles, to represent multiple solutions to a task. The representation of multiple versatile solutions is achieved though a hierarchical policy, which consists of a gating network and multiple sub-policies. We show that a naive implementation of a hierarchical policy is insufficient as it does not find distinct solutions. To address this problem, we introduce an entropy-based constraint which ensures that the agent finds distinct solutions with different sub-policies. Having a hierarchical policy based on multiple sub-policies additionally allows us to solve tasks that are non-linear using multiple linear sub-policies.

We evaluated the proposed method on two real robot tasks and several simulated tasks. The evaluations showed that our framework can be used to learn temporally extended sub-policies, also called options, and to sequence these sub-policies to learn complicated tasks on real systems.

The results show, that HiREPS is able to learn multiple solutions for complicated tasks and that HiREPS is able to learn piecewise linear solutions for non-linear tasks. The comparison to the non-hierarchical REPS method shows that HiREPS additionally often learns faster than REPS. This effect is especially intriguing as HiREPS does not only aim to learn one solution but multiple solutions at the same time and effectively uses fewer samples per sub-policy.

While the presented approach is able to learn a multi-modal policy using weights generated by solving one optimization problem, alternative solutions are possible. In the presented approach, computing the lower bound and, subsequently, solving the optimization problem and fitting the sub-policies is equivalent to an EM approach with only one iteration. While EM approaches typically benefit from multiple iterations, we observed no further benefit from applying multiple passes in our experiments. The observation that one EM iteration is sufficient can be explained by the fact that we start with a distribution which is highly similar to the target distribution. Thus, using HiREPS with $\kappa \gg 1$ results in a version of the REPS algorithm which works on a mixture model. Furthermore, alternative mixture model fitting approaches could be used to incorporate the entropy bound which is integrated into the optimization problem in the proposed approach. For example, Graça et al. (2007, 2009) show how EM can be performed with additional constraints such as, for example, the entropy constraint presented in this paper.

In this paper we focus on learning multiple sub-policies and do not solve the temporal extension aspect of the options framework directly. Temporal extension is inherently achieved in many of the presented experiments through the use of movement primitives. Using movement primitives, the time horizon is usually pre-determined. However, this time horizon could be made adaptive by including the duration of the movement primitive into the set of parameters that is learned by the agent. Learning both when to terminate options as well as learning how to construct options from atomic state-action pairs in a unified framework is an important aspect for future work. Equally, extending the framework to not only allow pruning of sub-policies, but also generating new sub-policies during the learning process could be an important addition to the versatility of the presented framework. In the presented version, options might be separated in the state-action space in the beginning of learning when actually multi-modalities exist on a smaller scale than can initially be detected. In such cases, it would be helpful to split one sub-policy into two sub-policies that can adapt to the individual modes.

Acknowledgments

The authors acknowledge the support of the European Union project # FP7-ICT-270327 (ComPLACS). This project has also received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No. 645582 (RoMaNS) as well as funding from the DFG project LearnRobotS.

References

M. Gheshlaghi Azar, V. Gómez, and H. J. Kappen. Dynamic Policy Programming. *Journal of Machine Learning Research*, 13(Nov):3207–3245, 2012.

J. A. Bagnell and J. C. Schneider. Covariant Policy Search. In *Proceedings of the IEEE International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

J. A. Bagnell and J. G. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proceedings of the IEEE International Conference for Robotics and Automation (ICRA)*, pages 1615–1620, 2001.

A.G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of the International Conference on Developmental Learning (ICDL),*, 2004.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

S. Calinon, P. Kormushev, and D. Caldwell. Compliant Skills Acquisition and Multi-Optima Policy Search with EM-based Reinforcement Learning. *Robotics and Autonomous Systems (RAS)*, 61 (4):369 – 379, 2013.

B. Da Silva, G. Konidaris, and A.G. Barto. Learning parameterized skills. *In Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012a.

C. Daniel, G. Neumann, and J. Peters. Learning Concurrent Motor Skills in Versatile Solution Spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012b.

C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Learning Sequential Motor Tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

P. Dayan and G. E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/neco.1997.9.2.271.

M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. KIT Scientific Publishing, 2010. ISBN 978-3-86644-569-7.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 2000.

G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot. *International Journal of Robotics Research*, 2008.

M. Ghavamzadeh and S. Mahadevan. Hierarchical Policy Gradient Algorithms. In *International Conference for Machine Learning (ICML)*, pages 226–233. AAAI Press, 2003.

J. V. Graça, K. Ganchev, and B. Taskar. Expectation maximization and posterior constraints. *Advances in Neural Information Processing Systems (NIPS)*, 2007.

J. V. Graça, K. Ganchev, B. Taskar, and F. Pereira. Posterior vs parameter sparsity in latent variable models. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*. 2009.

N. Hansen, S.D. Muller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

A. Ijspeert and S. Schaal. Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems 15*, (NIPS). MIT Press, Cambridge, MA, 2003.

J. Kober, B. J. Mohler, and J. Peters. Learning Perceptual Coupling for Motor Primitives. In *Intelligent Robots and Systems (IROS)*, pages 834–839, 2008.

J. Kober, K. Mülling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement Templates for Learning of Hitting and Batting. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010a.

J. Kober, E. Oztop, and J. Peters. Reinforcement Learning to adjust Robot Movements to New Situations. In *Proceedings of the Robotics: Science and Systems Conference (RSS)*, 2010b.

N. Kohl and Peter Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Proceedings of the IEEE International Conference for Robotics and Automation (ICRA)*, 2003.

G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1015–1023, 2009.

G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the fourier basis. *AAAI Conference on Artificial Intelligence (AAAI*, 2011.

P. Kormushev, S. Calinon, and D. Caldwell. Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

A. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-Efficient Contextual Policy Search for Robot Movement Skills. *Journal of Artificial Intelligence*, 2014.

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1071–1079, 2014.

K. Y. Levy and N. Shimkin. Unified inter and intra options learning using policy gradient methods. In *Recent Advances in Reinforcement Learning*, pages 153–164. Springer, New York City, 2012.

J. Morimoto and K. Doya. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51, 2001.

K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research*, (3):263–279, 2013.

G. Neumann. Variational Inference for Policy Search in Changing Situations. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

G. Neumann and J. Peters. Fitted Q-Iteration by Advantage Weighted Regression. In *Advances in Neural Information Processing Systems (NIPS)*. MA: MIT Press, 2009.

J. Peters and S. Schaal. Policy Gradient methods for Robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics Systems (IROS)*, Beijing, China, 2006.

J. Peters, K. Mülling, and Y. Altun. Relative Entropy Policy Search. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, 2010.

K. Rawlik, M. Toussaint, and S. Vijayakumar. On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference. In *Proceedings of Robotics: Science and Systems*, 2012.

M. T. Rosenstein. Robot Weightlifting by Direct Policy Search. *Proceedings of the International Joint Conference on Artificial Intelligence (ICJAI)*, 2001.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning Movement Primitives. In *Proceedings of the International Symposium on Robotics Research*, (ISRR), 2003.

J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

S. Still and D. Precup. An Information-theoretic Approach to Curiosity-driven Reinforcement Learning. *Proceedings of the International Conference on Humanoid Robotics*, 2011.

P. Stone. Scaling reinforcement learning toward RoboCup soccer. *Proceedings of the Conference on Machine Learning (ICML)*, 2001.

M. J. A. Strens. Policy Search using Paired Comparisons. *Journal of Machine Learning Research (JMLR)*, 3:921–950, 2003.

F. Stulp and S. Schaal. Hierarchical Reinforcement Learning with Movement Primitives. In *IEEE International Conference on Humanoid Robots (HUMANOIDS)*, 2012.

F. Stulp and O. Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. *Journées Francophones Planification, Décision, et Apprentissage pour la conduite de systèmes*, 2013.

R. S. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. In *Proccedings of the International Conference on Machine Learning (ICML)*, 1998.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 1999a.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112:181–211, 1999b.

R.S. Sutton. Generalization in Reinforcement Learning: Successful Examples using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Boston, MA, 1998.

E.. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11:3137–3181, 2010.

P. S Thomas and A. G. Barto. Motor primitive discovery. In *IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, 2012.

M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (po) mdps. *University of Edinburgh, School of Informatics Research Report EDI-INF-RR-0934*, 2006.

A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. *IEEE Transactions on Robotics*, 5, October 2010. ISSN 1552-3098.

H. van Hoof, J. Peters, and G. Neumann. Learning of non-parametric control policies with high-dimensional state features. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

D. Wingate, N. D. Goodman, D. M Roy, L. P. Kaelbling, and J. B. Tenenbaum. Bayesian policy search with policy priors. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Appendix A. Derivation of the Lower Bound

Consider the optimization problem in Equation (14) with the real conditional $p(o|s, a)$ instead of the responsibilities $\tilde{p}(o|s, a)$. For simplicity, we do not introduce the steady-state distribution and the normalization constraint as, our results are not affected by these constraints. The Lagrangian of this problem is then given by

$$\begin{aligned} F(\pi, \eta, \xi) = &\sum_{o \in \mathcal{O}} \iint \mu^{\pi}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \pi(o|\boldsymbol{s}) R_{\text{sa}} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \\ &+ \eta \left(\epsilon - \sum_{o \in \mathcal{O}} \iint \mu^{\pi}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \pi(o|\boldsymbol{s}) \log \frac{\mu^{\pi}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \pi(o|\boldsymbol{s})}{q(\boldsymbol{s}, \boldsymbol{a}) p(o|\boldsymbol{s}, \boldsymbol{a})} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \right) \\ &+ \xi \left(\tilde{\kappa} + \iint p(\boldsymbol{s}, \boldsymbol{a}) \sum_{o \in \mathcal{O}} p(o|\boldsymbol{s}, \boldsymbol{a}) \log p(o|\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \right). \end{aligned} \tag{26}$$

Simplifying the terms, we obtain

$$F(\pi, \eta, \xi) = \sum_{o \in \mathcal{O}} \iint \mu^{\pi}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \pi(o|\boldsymbol{s}) \Big(R_{\text{sa}} - \eta \log \frac{p(\boldsymbol{s}, \boldsymbol{a}, o)}{q(\boldsymbol{s}, \boldsymbol{a}) p(o|\boldsymbol{s}, \boldsymbol{a})^{1+\xi/\eta}} \Big) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} + \eta\epsilon + \xi\tilde{\kappa}. \tag{27}$$

However, determining a closed form solution for $p(\boldsymbol{s}, \boldsymbol{a}, o)$ is infeasible as the conditional $p(o|\boldsymbol{s}, \boldsymbol{a})$ is inside the log. Yet, we can determine a lower bound $L(p, \eta, \xi, \tilde{p})$ by using a proposal distribution $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})$ for $p(o|\boldsymbol{s}, \boldsymbol{a})$ which we can iteratively maximize in an Expectation-Maximization like manner. We need to verify that L is a lower bound on F and that maximizing L w.r.t $\tilde{p}$ is equivalent to setting $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a}) = p(o|\boldsymbol{s}, \boldsymbol{a})$, both of which follows from the relation

$$L = F - (\eta + \xi) \iint p(\boldsymbol{s}, \boldsymbol{a}) \underbrace{\sum_{o \in \mathcal{O}} p(o|\boldsymbol{s}, \boldsymbol{a}) \log \frac{p(o|\boldsymbol{s}, \boldsymbol{a})}{\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})}}_{D_{KL}\left(p(o|\boldsymbol{s}, \boldsymbol{a}) || \tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})\right) \geq 0} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}.$$

After the Expectation step, the lower bound is tight, i.e., $\max_{\tilde{p}} L(p, \eta, \xi, \tilde{p}) = F(p, \eta, \xi)$. In the Maximization step, we fix $\tilde{p}$ and maximize L w.r.t p, η and ξ. This combination defines our constraint optimization problem.

A.1 Derivation of Contextual HiREPS

We first reiterate the complete optimization problem, i.e.,

$$\begin{aligned}
\max_{\pi,\mu^\pi} J(\pi) = \max_{\pi,\mu^\pi} \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\mathcal{R}_{\boldsymbol{sa}}\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a},\\
\text{s. t.}\quad \epsilon \geq \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \log \frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})}\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a},\\
\tilde{\kappa} \geq -\sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)p(o|\boldsymbol{s},\boldsymbol{a}) \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a},\\
\hat{\boldsymbol{\phi}} = \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a},\\
1 = \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a},
\end{aligned}\tag{28}$$

which includes an additional constraint to represent prior knowledge about the desired behavior of sub-policies, i.e., that sub-policies should spread out and not overlap. In the entropy constraint, we replace only the responsibility term inside the log with the approximation $\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})$, as we can combine the expectation over the entropy into $\mathbb{E}_{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})} \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a})$.

The Lagrangian formulation reads

$$\begin{aligned}
L = \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \Big[\mathcal{R}_{\boldsymbol{sa}} - \eta \log \frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})}\\
+\xi \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a}) - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}) - \lambda\Big]\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a} + \eta\epsilon + \lambda + \boldsymbol{\theta}^T\hat{\boldsymbol{\phi}} + \xi\tilde{\kappa},
\end{aligned}\tag{29}$$

where η , $\boldsymbol{\theta}$, ξ and λ are Lagrangian parameters. To arrive at a closed form update rule for $\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})$, we differentiate L

$$\begin{aligned}
\frac{\mathrm{d}L}{\mathrm{d}\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})} &= \mathcal{R}_{\boldsymbol{sa}} - \eta \log \frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})} + \xi \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a}) - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}) - \lambda - \eta,\\
&= \mathcal{R}_{\boldsymbol{sa}} - \eta \log \frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}) - \lambda - \eta,
\end{aligned}\tag{30}$$

with

$$\xi \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a}) = \eta\frac{\xi}{\eta} \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a}) = \eta \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{\frac{\xi}{\eta}} = -\eta \log \frac{1}{\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{\xi/\eta}}.$$

We set the derivative to zero and write $V(\boldsymbol{s}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$ to solve for the update rule

$$\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) = q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta} \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} - V(\boldsymbol{s})}{\eta}\right) \exp\left(-1 - \frac{\lambda}{\eta}\right). \tag{31}$$

Here, we use the fact that

$$\sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a} = 1, \tag{32}$$

and, thus,

$$\exp\left(-1-\frac{\lambda}{\eta}\right)^{-1} = \sum_{o\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right)\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}. \tag{33}$$

Hence, the term $\exp\left(-1-\lambda/\eta\right)$ acts as a normalization constant and $\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})$ can be written as

$$\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) = \frac{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right)}{\sum_{o\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right)\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}}.$$

However, for improved clarity, we resort to writing the update equation in the proportional formulation

$$\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \propto q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right), \tag{34}$$

which depends only on the Lagrangian parameters ξ, η and $\boldsymbol{\theta}$ with $V(\boldsymbol{s}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$, but not on the parameter λ. The values for these Lagrangian parameters can be calculated by optimizing the dual problem. The dual formulation can be obtained by inserting Eq. (31) into the original Lagrangian formulation, i.e.,

$$\begin{aligned} g(\eta,\boldsymbol{\theta},\xi,\lambda) = \sum_{o\in\mathcal{O}}\iint &\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\Big[\mathcal{R}_{\boldsymbol{sa}} \\ &-\eta\log\frac{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right)\exp\left(-1-\frac{\lambda}{\eta}\right)}{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}} \\ &-\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})-\lambda\Big]\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}+\eta\epsilon+\lambda+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa}, \end{aligned} \tag{35}$$

which can easily be simplified to

$$g(\eta,\boldsymbol{\theta},\xi,\lambda) = \eta\epsilon+\lambda+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa}+\sum_{o\in\mathcal{O}}\iint\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})\eta\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}, \tag{36}$$

where we again use Eq. (32) to simplify further to

$$g(\eta,\boldsymbol{\theta},\xi,\lambda) = \eta\epsilon+\lambda+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa}+\eta. \tag{37}$$

At this point we rewrite Eq. (33) as

$$\eta+\lambda = \eta\log\sum_{o\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right)\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}, \tag{38}$$

where we used $\log(a^{-1}) = -\log(a)$ and multiplied by η. We insert Eq. (38) back into Eq. (36) and get

$$\begin{aligned} g(\eta,\boldsymbol{\theta},\xi) =&\eta\log\sum_{o\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}-V(\boldsymbol{s})}{\eta}\right) \\ &+\eta\epsilon+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa}. \end{aligned} \tag{39}$$

The dual function $g(\eta, \boldsymbol{\theta}, \xi)$ is formulated in such a way that it does not depend on the Lagrangian parameter λ, which acts as a normalization factor.

If we work on samples $\{\boldsymbol{s}, \boldsymbol{a}\}_i$ with $i = 1, 2, \dots N$, we have to divide by the generating distribution $q(\boldsymbol{s}, \boldsymbol{a})$ and obtain

$$\begin{aligned} g(\eta, \boldsymbol{\theta}, \xi) =& \eta \log \frac{1}{N} \sum_{o \in \mathcal{O}} \sum_{i=1}^{N} \tilde{p}(o|\boldsymbol{s}_i, \boldsymbol{a}_i)^{1+\xi/\eta} \exp\left(\frac{R_i - V(\boldsymbol{s}_i)}{\eta}\right) \\ &+ \eta\epsilon + \boldsymbol{\theta}^T \hat{\boldsymbol{\phi}} + \xi\tilde{\kappa}, \end{aligned} \tag{40}$$

which we can optimize using standard optimization libraries.

A.2 Derivation of Skill Sequencing

As above, we first reiterate the complete optimization problem, i.e.,

$$\begin{aligned} \max_{\pi_k, \mu_k^\pi} J(\pi_{1:K}) = \max_{\pi, \mu_k^\pi} \sum_{k=1}^{K} \sum_{o \in \mathcal{O}} & \iint \mathcal{R}_{\boldsymbol{sa}}^k \pi_k(\boldsymbol{a}|\boldsymbol{s}) \pi_k(o|\boldsymbol{s}) \mu_k^\pi(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} + \int \mu_{K+1}^\pi(\boldsymbol{s}) \mathcal{R}_{\boldsymbol{s}}^{K+1} \, \mathrm{d}\boldsymbol{s}, \\ \mathrm{s.\,t.} : \epsilon \geq & \int \mu_{K+1}^\pi(\boldsymbol{s}) \log \frac{\mu_{K+1}^\pi(\boldsymbol{s})}{q_{K+1}(\boldsymbol{s})} \, \mathrm{d}\boldsymbol{s}, \\ \hat{\boldsymbol{\phi}}_0 = & \int \boldsymbol{\phi}(\boldsymbol{s}') \mu_1^\pi(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}', \\ \forall k \leq K : \epsilon \geq & \sum_{o \in \mathcal{O}} \iint \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o) \pi_k(o|\boldsymbol{s}) \log \frac{\mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o) \pi_k(o|\boldsymbol{s})}{q_k(\boldsymbol{s}, \boldsymbol{a}) \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a})} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \\ \tilde{\kappa} \geq & -\sum_{o \in \mathcal{O}} \iint \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o) p_k(\boldsymbol{s}, \boldsymbol{a}) \pi_k(o|\boldsymbol{s}) \log \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \\ \int \boldsymbol{\phi}(\boldsymbol{s}') \mu_{k+1}^\pi(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' = & \sum_{o \in \mathcal{O}} \iiint \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o) \pi_k(o|\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{s}' \, \mathrm{d}\boldsymbol{a}, \\ \forall k : 1 = & \textstyle\sum_{o \in \mathcal{O}} \displaystyle\iint \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}, o|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \end{aligned} \tag{41}$$

The Lagrangian formulation reads

$$\begin{aligned} L = \boldsymbol{\theta}_1^T \hat{\boldsymbol{\phi}}_0 + \sum_{k=1}^{K} \eta_k \epsilon + \lambda_k + \int \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}) \mu_k^\pi(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} + \xi_k \tilde{\kappa} + \sum_{o \in \mathcal{O}} \iint \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o) \pi_k(o|\boldsymbol{s}) \\ \left[\mathcal{R}_{\boldsymbol{sa}}^k - \eta_k \log \frac{\mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{a}|\boldsymbol{s}, o) \pi_k(o|\boldsymbol{s})}{q_k(\boldsymbol{s}, \boldsymbol{a}) \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a})} + \xi_k \log \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a}) - \int \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' - \lambda_k\right] \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \\ + \int \mu_{K+1}^\pi(\boldsymbol{s}) \left[\mathcal{R}_{\boldsymbol{s}}^{K+1} - \eta_{K+1} \log \frac{\mu_{K+1}^\pi(\boldsymbol{s})}{q_{K+1}(\boldsymbol{s})} - \lambda_{K+1}\right] \mathrm{d}\boldsymbol{s} + \eta_{K+1} \epsilon + \lambda_{K+1}, \end{aligned} \tag{42}$$

where η_k , $\boldsymbol{\theta}_k$, ξ_k and λ_k are Lagrangian parameters. To arrive at a closed form update rule for $\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})$, we differentiate L

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})} =&\mathcal{R}_{\boldsymbol{sa}}^k - \eta_k \log \frac{\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})}{q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})} + \xi_k \log \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a}) \\ &+ \int \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' - \boldsymbol{\theta}_k^T\boldsymbol{\phi}(\boldsymbol{s}) - \lambda_k - \eta_k, \\ =&\mathcal{R}_{\boldsymbol{sa}}^k - \eta_k \log \frac{\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})}{q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k}} \\ &+ \int \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' - \boldsymbol{\theta}_k^T\boldsymbol{\phi}(\boldsymbol{s}) - \lambda_k - \eta_k, \end{aligned} \quad (43)$$

with

$$\xi_k \log \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a}) = \eta_k \frac{\xi_k}{\eta_k} \log \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a}) = \eta_k \log \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{\xi_k/\eta_k} = -\eta_k \log \frac{1}{\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{\xi_k/\eta_k}}$$

and set the derivative to zero to solve for the update rule

$$\begin{aligned}\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s}) =&q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k} \\ &\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}^k + \int \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' - \boldsymbol{\theta}_k^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta_k}\right) \exp\left(-1 - \frac{\lambda_k}{\eta_k}\right). \end{aligned} \quad (44)$$

For improved readability we write $\mathbb{E}[V_{k+1}(\boldsymbol{s}')] = \int \mathcal{P}_{\boldsymbol{ss}'}^{\boldsymbol{a}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'$ and $V_k(\boldsymbol{s}) = \boldsymbol{\theta}_k^T\boldsymbol{\phi}(\boldsymbol{s})$. As before, we use

$$\sum_{o\in\mathcal{O}} \iint \mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a},o|\boldsymbol{s})\pi_k(o|\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a} = 1, \quad (45)$$

and, thus, obtain

$$\begin{aligned}\exp\left(-1 - \frac{\lambda_k}{\eta_k}\right)^{-1} =&\sum_{o\in\mathcal{O}} \iint q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k} \\ &\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}^k + \mathbb{E}[V_{k+1}(\boldsymbol{s}')] - V_k(\boldsymbol{s})}{\eta_k}\right)\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}. \end{aligned} \quad (46)$$

Hence, the term $\exp\left(-1 - \frac{\lambda_k}{\eta_k}\right)$ acts as a normalization constant and $\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})$ can be written as

$$\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s}) = \frac{q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}^k + \mathbb{E}[V_{k+1}(\boldsymbol{s}')] - V_k(\boldsymbol{s})}{\eta_k}\right)}{\sum_{o\in\mathcal{O}} \iint q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}^k + \mathbb{E}[V_{k+1}(\boldsymbol{s}')] - V_k(\boldsymbol{s})}{\eta_k}\right)\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}}.$$

For improved clarity, we resort to writing the update equation in the proportional formulation

$$\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s}) \propto q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k}\exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}^k + \mathbb{E}[V_{k+1}(\boldsymbol{s}')] - V_k(\boldsymbol{s})}{\eta}\right), \quad (47)$$

which depends only on the Lagrangian parameters ξ, η and $\boldsymbol{\theta}$ with $V(\boldsymbol{s}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$, but not on the parameter λ. The values for these Lagrangian parameters can be calculated by optimizing the dual problem.

As before, we set the solution for $\mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})$ back into the Lagrangian and simplify to get a dual function that does not depend on the Lagrangian parameters λ_k, i.e.,

$$g(\eta_{1:K+1}, \boldsymbol{\theta}_{1:K+1}, \xi_{1:K+1}) = \boldsymbol{\theta}_1^T\hat{\boldsymbol{\phi}}_0 + \sum_{k=1}^{K+1} \eta_k\epsilon + \xi_k\tilde{\kappa} + \eta_k \log \sum_{o\in\mathcal{O}} \iint q_k(\boldsymbol{s},\boldsymbol{a})\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi_k/\eta_k} \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}}^k + \mathbb{E}[V_{k+1}(\boldsymbol{s}')] - V_k(\boldsymbol{s})}{\eta_k}\right) \mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}. \tag{48}$$

Now, the dual function $g(\eta_{1:K+1}, \boldsymbol{\theta}_{1:K+1}, \xi_{1:K+1})$ is formulated in such a way that it does not depend on the Lagrangian parameters λ_k, which acts as a normalization factor.

A.3 Derivation of Infinite Horizon HiREPS

The derivation of the infinite horizon case follows the derivations given above, where the Lagrangian is given by

$$L = \sum_{o\in\mathcal{O}} \iint \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \Big[\mathcal{R}_{\boldsymbol{sa}} - \eta\log\frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})} + \xi\log\tilde{p}(o|\boldsymbol{s},\boldsymbol{a}) - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}) - \lambda\Big] \mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a} + \eta\epsilon + \lambda + \int \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' + \xi\tilde{\kappa}, \tag{49}$$

and the resulting update rule and dual function are given as

$$\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o)\pi(o|\boldsymbol{s}) \propto q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta} \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})}{\eta}\right), \tag{50}$$

and

$$g(\eta, \boldsymbol{\theta}, \xi) = \eta\log\sum_{o\in\mathcal{O}} \iint q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta} \exp\left(\frac{\mathcal{R}_{\boldsymbol{sa}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})}{\eta}\right) + \eta\epsilon + \xi\tilde{\kappa}. \tag{51}$$

As for the other cases we can find the minimum of the dual function using standard optimization libraries.