Noname manuscript No.
(will be inserted by the editor)

Probabilistic Inference for Determining Options in Reinforcement Learning

Christian Daniel · Herke van Hoof · Jan Peters · Gerhard Neumann

Received: date / Accepted: date

Abstract Tasks that require many sequential decisions or complex solutions are hard to solve using conventional reinforcement learning algorithms. Based on the semi Markov decision process setting (SMDP) and the option framework, we propose a model which aims to alleviate these concerns. Instead of learning a single monolithic policy, the agent learns a set of simpler sub-policies as well as the initiation and termination probabilities for each of those sub-policies. While existing option learning algorithms frequently require manual specification of components such as the sub-policies, we present an algorithm which infers all relevant components of the option framework from data. Furthermore, the proposed approach is based on parametric option representations and works well in combination with current policy search methods, which are particularly well suited for continuous real-world tasks. We present results on SMDPs with discrete as well as continuous state-action spaces. The results show that the presented algorithm can combine simple sub-policies to solve complex tasks and can improve learning performance on simpler tasks.

1 Introduction

Solving tasks which require long decision sequences or complex policies is an important challenge in reinforcement learning (RL). The option framework [38] is a promising approach to simplify the complexity of such tasks. In the option framework, a reinforcement learning (RL) agent can choose between actions and macro-actions, which are carried out over multiple time steps [28, 38].

C. Daniel [1,2] · H. van Hoof[1] · J. Peters[1,3] · G. Neumann[1]
E-mail: Christian.Daniel@de.bosch.com

[1] Technische Universität Darmstadt
Hochschulstrasse 10, 64289 Darmstadt, Germany

[2] Bosch Corporate Research, Cognitive Systems
Robert-Bosch-Campus 1, 71272 Renningen, Germany

[3] Max-Planck-Institut für Intelligente Systeme
Speemannstraße 38, 72076 Tübingen, Germany

Using these macro-actions, the agent has to make less decisions to solve a task. Furthermore, even if macro-actions are based on simple policies, the combination of multiple macro-actions can represent more complex solutions than the simple policies would allow for on their own. For example, if a given task requires a nonlinear policy, a combination of multiple linear sub-policies might still be able to solve this task. Such an automated decomposition of complex solutions can simplify the learning problem in many domains.

Based on the SMDP setting [30], the option framework [37] incorporates such macro actions. An option consists of a sub-policy, an initiation set and a termination probability. After an option is initiated, actions are generated by the sub-policy until the option is terminated and a new option is activated. While the option framework has received considerable attention [28, 38, 8], to date most algorithms either require the manual specification of the activation policies or sub-policies. Algorithms for autonomous option discovery typically depend on discrete state-action spaces [20, 18, 34] with exceptions such as the work of Konidaris et al. [14]. Furthermore, many existing algorithms first explore a given MDP and learn suitable options afterwards [22, 33]. Hence, they are not aimed at leveraging the efficiency of options in the initial stages of learning but rather aim at transferring the options to new tasks. These approaches are powerful in scenarios where options can be transferred to similar tasks in the future. In contrast, the approach suggested in this paper aims at directly learning suitable options for the problem at hand while also being applicable in continuous state-action spaces.

In continuous state-action spaces, policy search (PS) methods which optimize parametrized policies have been shown to learn efficiently in simulated and real world tasks [24, 13]. Thus, the compatibility of the proposed option discovery framework with PS methods such as PoWER [13] and REPS [29] is an important goal of this paper. In the discrete setting, the framework can equally be combined with a wide range of methods such as as Q-Learning [41] and LSPI [16]. Furthermore, many complex tasks can be solved through combinations of simple behavior patterns. The proposed framework can combine multiple simple sub-policies to achieve complex behavior if a single sub-policy would be insufficient to solve the task. These simpler sub-policies are easier to learn which can improve the overall learning speed.

To arrive at a general framework which works in discrete and continuous settings and requires minimal prior knowledge, we propose a probabilistic formulation of the option framework where all components are represented by distributions. To learn these options from data, we propose a probabilistic inference algorithm based on the expectation maximization algorithm [2] to determine the options' components. The resulting algorithm offers three key benefits

1. It infers a data-driven segmentation of the state-space to learn the initialization and termination probability for each option.
2. It is applicable to discrete as well as continuous state-action spaces.
3. It outperforms monolithic algorithms on discrete tasks and can solve complex continuous tasks by combining simpler sub-policies.

Together, these contributions allow for learning of options from data with minimal prior knowledge.

1.1 Problem Statement

We consider the option framework within the MDP setting with states $\boldsymbol{s} \in \mathcal{S}$, actions $\boldsymbol{a} \in \mathcal{A}$, option indices $o \in \mathcal{O}$, and termination events $b \in \mathcal{B}$. Every option consists of a sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and a termination policy $\pi(b|\boldsymbol{s}, \bar{o})$, where b encodes a binary termination event and $\bar{o}$ is the index of the previously active option. Furthermore, one global activation policy $\pi(o|\boldsymbol{s})$ governs the activation of options following a termination event. This activation policy $\pi(o|\boldsymbol{s})$ replaces the initiation set used in classical algorithms and defines a probability of initiating an option in a given state. A new option can only be activated if the previously active option $\bar{o}$ is terminated. Thus, the agent uses the same option for multiple time steps if it is not terminated. The option transition model is given as

$$p(o|\boldsymbol{s}, b, \bar{o}) = \begin{cases} \pi(o|\boldsymbol{s}) & \text{if } b = 1, \\ \delta_{o=\bar{o}} & \text{if } b = 0, \end{cases} \tag{1}$$

and ensures that option $\bar{o}$ remains active until a termination event occurs. After each termination, a new option will be sampled according to $\pi(o|\boldsymbol{s})$.

In Section 2, we show how to learn a hierarchical policy defined by the options given a set of demonstrated trajectories, i.e., how to solve the imitation learning problem. We formulate the option framework as a graphical model where the index of the executed option is treated as latent variable. Subsequently, we show how an EM algorithm can be used to infer the parameters of the option components. In Section 3 we extend the imitation learning solution to allow for reinforcement learning, i.e., for iteratively improving the hierarchical policy such that it maximizes a reward function.

2 Learning Options From Data

The goal of this paper is to determine sub-policies, termination policies and the activation policy from data with minimal prior knowledge. All option components are represented by parametrized distributions, governed by the parameter vector $\boldsymbol{\theta} = \{\boldsymbol{\theta}_A, \boldsymbol{\theta}_O, \boldsymbol{\theta}_B\}$. The individual components are given as binary classifiers for the termination policies for each option $\pi(b|\boldsymbol{s}, o = i; \boldsymbol{\theta}_B^i)$, one global multi-class classifier for the activation policies $\pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O)$ and the individual sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s}, o = i; \boldsymbol{\theta}_A^i)$. We use the notation $\pi(\cdot)$ to denote option components, which we will ultimately aim to learn and $p(\cdot)$ for all other distributions. Our goal is to estimate the set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_A, \boldsymbol{\theta}_O, \boldsymbol{\theta}_B\}$, which explain one or more demonstrated trajectories $\tau = \{\tau_1, \ldots, \tau_T\}$ with $\tau_t = \{\boldsymbol{s}_t, \boldsymbol{a}_t\}$. Crucially, the observations τ do not contain the option indices o nor the termination events b but only states $\boldsymbol{s}$ and actions $\boldsymbol{a}$. Both, the option index and the termination events are latent variables. While the proposed method learns all option components from data, it requires a manual selection of the total number of desired options. Future work could replace this requirement by, for example, sampling the number of options through a Dirichlet process.

For this Section, we ignore the optimality of the resulting trajectories and focus on the imitation learning problem, i.e., how to recover hierarchical policies from trajectory observations. We propose a probabilistic option framework, where all

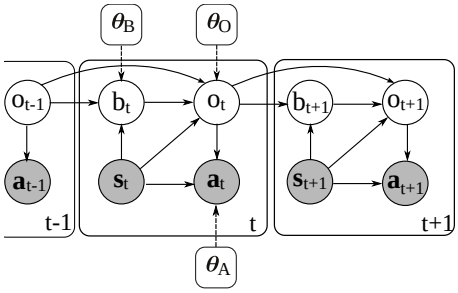


Fig. 1: The graphical model of the proposed framework. The termination policy $\pi(b|\boldsymbol{s}, o)$ decides whether to keep executing the previously active option or to terminate its execution. After a termination event, the activation policy $\pi(o|\boldsymbol{s})$ samples a new option and the sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ samples an action. Arrows from $\boldsymbol{s}_t, \boldsymbol{a}_t$ to $\boldsymbol{s}_{t+1}$ are not shown to reduce clutter as we do not aim to model the transition probability distribution. Shaded nodes indicate observed variables, while clear nodes indicate the hidden variables. The model parameters $\boldsymbol{\theta}_B, \boldsymbol{\theta}_O, \boldsymbol{\theta}_A$ are shown in rounded square boxes.

option components are represented as distributions. We can then recover distributions over the latent variables using probabilistic inference techniques.

2.1 The Graphical Model for Options

To apply these probabilistic inference techniques, we need to specify how the variables in our model interact with each other. Figure 1 shows a graphical model of the proposed setting and the resulting hierarchical policy can be given as

$$\pi(\boldsymbol{a}|\boldsymbol{s}, \bar{o}; \boldsymbol{\theta}) = \sum_{o \in \mathcal{O}} \sum_{b \in \mathcal{B}} \pi(b|\boldsymbol{s}, \bar{o}; \boldsymbol{\theta}_B)\pi(o|\boldsymbol{s}, b, \bar{o}; \boldsymbol{\theta}_O)\pi(\boldsymbol{a}|\boldsymbol{s}, o; \boldsymbol{\theta}_A). \tag{2}$$

The graphical model in Fig 1 shows that the following operations occur in every time step. First, the termination policy $\pi(b|\boldsymbol{s}, \bar{o}; \boldsymbol{\theta}_B)$ samples a termination event b. According to Equation (1) the previously active option $\bar{o}$ remains active if no termination occurs. Otherwise, the activation policy $\pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O)$ samples a new option index o based on the current state $\boldsymbol{s}$. Finally, the sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o; \boldsymbol{\theta}_A)$ samples an action $\boldsymbol{a}$ based on the state $\boldsymbol{s}$.

2.2 Expectation Maximization for Options

The graphical model for the option framework is a special case of a hidden Markov model (HMM). The Baum-Welch algorithm [2] is an EM algorithm for estimating the parameters of a HMM. We will now state the Baum-Welch algorithm for our special case of the option model, where we consider the special case of a single trajectory for improved clarity. The extension to multiple trajectories, however, is straightforward.

In our graphical model, the latent variables $\{o_{1:T}, b_{1:T}\}$ are given by trajectories of the option index and the termination event. EM algorithms optimize a lower-bound of the marginal log-likelihood

$$\log p(\tau|\boldsymbol{\theta}) \;\geq \sum_{o_{1:T}, b_{1:T}} p(o_{1:T}, b_{1:T}|\tau, \boldsymbol{\theta}^{\text{old}}) \log p(o_{1:T}, b_{1:T}, \tau|\boldsymbol{\theta}) = Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}). \quad (3)$$

The lower-bound is maximized iteratively. In the expectation (E-) Step, we compute the posterior probabilities of the latent variables using our current estimate of the parameters $\boldsymbol{\theta}^{\text{old}}$. In the maximization (M-) Step, we maximize $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ w.r.t $\boldsymbol{\theta}$ to obtain a new estimate of the parameter vector. As the lower-bound is tighter after every E-Step, the EM algorithm is guaranteed to converge to a local maximum of the marginal log-likelihood.

As we are dealing with time series, we can leverage the structure of the trajectory distribution

$$\begin{aligned} p(o_{1:T}, b_{2:T}, \tau|\boldsymbol{\theta}) \;&=\; \pi(o_1|\boldsymbol{s}_1; \boldsymbol{\theta}_O)\pi(a_1|o_1, \boldsymbol{s}_1; \boldsymbol{\theta}_A) \quad (4) \\ \prod_{t=1}^{T-1} \pi(b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}_B)\pi(o_{t+1}|\boldsymbol{s}_{t+1}, b_{t+1}, \boldsymbol{\theta}_O)\pi(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1}, o_{t+1}; \boldsymbol{\theta}_A)p(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t, \boldsymbol{s}_t). \end{aligned}$$

After substituting Eq. (4) into Eq. (3) and rearranging terms, the lower bound decomposes in a sum over the time steps, i.e.,

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = &\sum_{t=1}^{T-1} \sum_{o_t} \sum_{b_{t+1}} p(o_t, b_{t+1}|\tau; \boldsymbol{\theta}^{\text{old}}) \log \pi(b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}_B) \\ &+ \sum_{t=1}^{T} \sum_{o_t} p(o_t|\tau; \boldsymbol{\theta}^{\text{old}}) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t, o_t; \boldsymbol{\theta}_A), \\ &+ \sum_{t=1}^{T} \sum_{o_t} p(o_t, b_t = 1|\tau; \boldsymbol{\theta}^{\text{old}}) \log \pi(o_t|\boldsymbol{s}_t; \boldsymbol{\theta}_O). \quad (5) \end{aligned}$$

The posterior probabilities $p(o_t, b_{t+1}|\tau; \boldsymbol{\theta}^{\text{old}}), p(o_t|\tau; \boldsymbol{\theta}^{\text{old}})$ and $p(o_t, b_t = 1|\tau; \boldsymbol{\theta}^{\text{old}})$ can be recovered from two posterior belief distributions

$$\xi_t(o_{t:t+1}, b_{t:t+1}) = p(o_{t:t+1}, b_{t:t+1}|\tau; \boldsymbol{\theta}^{\text{old}}), \quad (6)$$

and

$$\gamma_t(o_t, b_t) = p(o_t, b_t|\tau; \boldsymbol{\theta}^{\text{old}}). \quad (7)$$

Consequently, we need to infer only the posteriors γ_t and ξ_t in the E-Step, and not the probability $p(o_{1:T}, b_{1:T}|\tau)$ of a whole trajectory.

2.2.1 Expectation Step.

During the expectation step, the responsibilities

$$\gamma_t(o_t, b_t) = z_t^{-1} \alpha_t(o_t, b_t) \beta_t(o_t, b_t), \tag{8}$$

and

$$\begin{aligned} \xi_t(o_{t:t+1}, b_{t:t+1}) = & z_t^{-1} \alpha_t(o_t, b_t) \beta_{t+1}(o_{t+1}, b_{t+1}) \\ & p(\boldsymbol{a}_{t+1}, o_{t+1}, b_{t+1} | \boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}^{\text{old}}), \end{aligned} \tag{9}$$

can be determined using forward messages $\alpha_t(o_t, b_t)$ and backward messages $\beta_t(o_t, b_t)$, where z_t is the normalization constant of $\gamma_t(o_t, b_t)$. Given the option model, the forward messages

$$\alpha_t(o_t, b_t) = p(\boldsymbol{a}_{1:t}, o_t, b_t | \boldsymbol{s}_{1:t}; \boldsymbol{\theta}^{\text{old}})$$

can be computed recursively by

$$\alpha_t(o_t, b_t) = \sum_{o_{t-1}} \sum_{b_{t-1}} \alpha_{t-1}(o_{t-1}, b_{t-1}) p(\boldsymbol{a}_t, b_t, o_t | \boldsymbol{s}_t, o_{t-1}; \boldsymbol{\theta}^{\text{old}}), \tag{10}$$

with $\alpha_1(o_1, b_1) = p(\boldsymbol{a}_1, b_1, o_1 | \boldsymbol{s}_1; \boldsymbol{\theta}^{\text{old}})$. Based on these forward messages, the backward messages

$$\beta_t(o_t, b_t) = p(\boldsymbol{a}_{t+1:T}, | \boldsymbol{s}_{t+1:T}, b_t, o_t; \boldsymbol{\theta}^{\text{old}}) \tag{11}$$

are computed recursively by

$$\beta_{t-1}(o_{t-1}, b_{t-1}) = \sum_{o_t} \sum_{b_t} \beta_t(o_t, b_t) p(\boldsymbol{a}_t, b_t, o_t | \boldsymbol{s}_t, o_{t-1}, b_{t-1}; \boldsymbol{\theta}^{\text{old}}), \tag{12}$$

with $\beta_T(o_T, b_T) = \mathbf{1}$.

2.2.2 Maximization Step.

Given the distributions over latent variables and the observed state-action samples, the parameters $\boldsymbol{\theta}$ can be determined by maximizing Equation (5). Since $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ is decoupled, independent optimization can be performed for the sub-policies, termination policies and the activation policy.

Termination Policies. For optimizing the termination models, we have to consider the first term of the lower bound which is given by

$$Q_B^i(\boldsymbol{\theta}_B, \boldsymbol{\theta}^{\text{old}}) = \sum_t \sum_{b_{t+1}} p(o_t = i, b_{t+1} | \tau; \boldsymbol{\theta}^{\text{old}}) \log \pi(b_{t+1} | \boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}_B) \tag{13}$$

for each time step t and a single option $o_t = i$. This term can be rewritten as

$$\begin{aligned} Q_B^i(\boldsymbol{\theta}_B, \boldsymbol{\theta}^{\text{old}}) = \sum_t w_{B,i}(t) \Big(& t_{B,i}(t) \log p(b_{t+1} = 1 | o_t = i; \boldsymbol{\theta}_B) + \\ & \big(1 - t_{B,i}(t)\big) \log \big(1 - p(b_{t+1} = 1 | o_t = i; \boldsymbol{\theta}_B)\big) \Big), \end{aligned} \tag{14}$$

where $w_{B,i}(t)$ is a weight and $t_{B,i}(t)$ defines the target probability of the termination event. Equation (14) resembles a weighted cost function for logistic regression [3] where the weights are given by

$$w_{B,i}(t) = p(o_t = i|\tau; \boldsymbol{\theta}^{\text{old}}) = \sum_{b_t} \gamma(o_t = i, b_t).$$

The target probability of the termination event is given by

$$t_{B,i}(t) = p(b_{t+1} = 1|o_t = i, \tau; \boldsymbol{\theta}^{\text{old}}) = \frac{\sum_{o_{t+1}, b_t} \xi_t(o_t = i, o_{t+1}, b_t, b_{t+1} = 1)}{p(o_t = i|\tau; \boldsymbol{\theta}^{\text{old}})}.$$

Using these target probabilities and weights, standard techniques can be used to fit, for example, a sigmoidal classifier for each termination policy [3].

Activation Policy. The case of the activation policy $\pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O)$ follows the argument of the termination policies. Similarly to the termination policies, we consider the relevant term of the lower bound

$$Q_O(\boldsymbol{\theta}_O, \boldsymbol{\theta}^{\text{old}}) = \sum_{t=1}^{T} \sum_{o_t} p(o_t, b_t = 1|\tau; \boldsymbol{\theta}^{\text{old}}) \log \pi(o_t|\boldsymbol{s}_t; \boldsymbol{\theta}_O), \tag{15}$$

to extract weights

$$w_O(t) = \ p(b_t = 1|\tau; \boldsymbol{\theta}^{\text{old}}) = \sum_{o} \gamma_t(o_t, b_t),$$

and target probabilities

$$t_O(t) = p(o_t|\tau, b_t = 1; \boldsymbol{\theta}^{\text{old}}) = \frac{\gamma_t(o_t, b_t)}{\sum_o \gamma_t(o_t, b_t)}.$$

While there exists an individual termination policy for each option, in our implementation, only a single global activation policy governs the initialization of all options. Thus, only one multi-class classifier has to be learned. Here, having one global classifier is a design decision and other alternatives are feasible. Given the weighted target probabilities of the activation policy, standard methods can be used to fit a multi-class classifier [3].

Sub-Policies. Finally, the sub-policies have to be fit. The relevant terms of the lower bound are given by

$$Q_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}^{\text{old}}) = \sum_{t=1}^{T} \sum_{o_t} p(o_t|\tau; \boldsymbol{\theta}^{\text{old}}) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t, o_t; \boldsymbol{\theta}_A), \tag{16}$$

such that the weights are given by

$$w_{A,i}(t) = \ p(o_t = i|\tau; \boldsymbol{\theta}^{\text{old}}) = \sum_{b_t} \gamma_t(o_t = i, b_t).$$

Here, it is worth noticing that the weights for the sub-policies are identical to the weights of the termination policies. However, the target values are different. For the sub-policies the target values are given directly by the observed actions $\boldsymbol{a}_t$, given the observed states $\boldsymbol{s}_t$. Given the weighted state-action pairs, stochastic policies, such as linear Gaussian policies, can be fit to the data.

Feature Representations of the State. The above derivations are given in their most general form, where each option component depends directly on the state $\boldsymbol{s}$. In practice, it may often be beneficial to train the individual components on a feature transformation $\boldsymbol{\phi}(\boldsymbol{s})$ of the state. Such features might, for example, be polynomial expansion of the state variable or a kernelized representation. When using feature representations, different representations can be chosen for the individual option components.

3 Probabilistic Reinforcement Learning for Option Discovery

The results in the previous section allow us to recover a hierarchical policy from state-action observations, i.e., to perform imitation learning with hierarchical policies. In this section, we extend these results to allow for reinforcement learning. Using the hierarchical policy, the agent aims to maximize the expected reward

$$J(\pi) = \mathbb{E}_\pi \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{17}$$

where γ is the discount factor. We will show how probabilistic inference based methods can be combined with EM algorithm of Section 2 to learn a reward maximizing hierarchical policy.

3.1 Probabilistic Reinforcement Learning Algorithms

There exist several algorithms which use probabilistic inference techniques for computing the policy update in reinforcement learning [7, 39, 13, 29]. More formally, they either re-weight state-action trajectories or state-action pairs according to the estimated quality of the state-action pair and, subsequently, use a weighted maximum likelihood estimate to obtain the parameters of a new policy π^*.

A common approach is to use an exponential transformation of the advantage function $A(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s})$, where $Q(\boldsymbol{s}_i, \boldsymbol{a}_i)$ is the Q-function and $V(\boldsymbol{s}_i)$ is the value function, to reweight the state-action distribution [29, 5]. The resulting desired state-action distribution $p(\boldsymbol{s}, \boldsymbol{a})$ is then given by

$$p(\boldsymbol{s}, \boldsymbol{a}) \propto q(\boldsymbol{s}, \boldsymbol{a}) \exp\left(\frac{Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s})}{\eta}\right),$$

where $q(\boldsymbol{s}, \boldsymbol{a})$ is the sampling distribution which is typically obtained by sampling from the old policy $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s})$. The parameter η is a temperature parameter that is either optimized by the algorithm [29, 5] or manually set [39, 13]. A new parametrized policy π^* can then be obtained by minimizing the expected Kulback-Leibler divergence between the re-weighted policy update $p(\boldsymbol{a}|\boldsymbol{s})$ and the

new parametric policy π^* [40], i.e.,

$$\begin{aligned}\pi^* &= \text{argmin}_\pi \mathbb{E}_{p(\boldsymbol{s})}\left[\text{KL}\left(p(\boldsymbol{a}|\boldsymbol{s})||\pi(\boldsymbol{a}|\boldsymbol{s})\right)\right] \\ &= \text{argmax}_\pi \int p(\boldsymbol{s},\boldsymbol{a}) \log \pi(\boldsymbol{a}|\boldsymbol{s}) \; \mathrm{d}\boldsymbol{s} \; \mathrm{d}\boldsymbol{a} + \text{const} \\ &= \text{argmax}_\pi \int q(\boldsymbol{s},\boldsymbol{a}) \exp\left(\frac{Q(\boldsymbol{s},\boldsymbol{a}) - V(\boldsymbol{s})}{\eta}\right) \log \pi(\boldsymbol{a}|\boldsymbol{s}) \; \mathrm{d}\boldsymbol{s} \; \mathrm{d}\boldsymbol{a} + \text{const} \\ &\approx \text{argmax}_\pi \sum_{(\boldsymbol{s}_i,\boldsymbol{a}_i)\sim q(\boldsymbol{s},\boldsymbol{a})} w(\boldsymbol{s}_i,\boldsymbol{a}_i) \log \pi(\boldsymbol{a}_i|\boldsymbol{s}_i), \end{aligned} \tag{18}$$

where

$$w(\boldsymbol{s}_i,\boldsymbol{a}_i) = \exp\left(\frac{Q(\boldsymbol{s}_i,\boldsymbol{a}_i) - V(\boldsymbol{s}_i)}{\eta}\right)$$

defines a weighting for each state action pair. It can now be easily seen that Equation 18 is equivalent to a weighted maximum likelihood estimate for the policy π^*. Specifically, we could use standard maximum likelihood techniques to fit, for example, a linear Gaussian policy to the observed state-action samples $\{\boldsymbol{s}_i, \boldsymbol{a}_i\}$, reweighted by the the weights $w(\boldsymbol{s}_i, \boldsymbol{a}_i)$, which would minimize the Kullback-Leibler divergence in Eq. (18).

We will employ different RL algorithms in the continuous and discrete setting to compute these weightings. For the continuous experiments, we will use the HiREPs [5] algorithm. While other algorithms are equally feasible [39, 13, 29], HiREPS is able to actively separate sub-policies if options are available by ensuring that different sub-policies generate distinct behaviors in similar states. Ensuring such versatility of the sub-policies is achieved by constraining the entropy of responsibilities for each option, i.e.,

$$H\big(p(\boldsymbol{a}|\boldsymbol{s},o)\big) > \kappa, \tag{19}$$

where κ is manually set. Furthermore, HiREPS limits the KL divergence of state-action distributions induced by policy updates such that

$$\text{KL}\big(p(\boldsymbol{s},\boldsymbol{a})||q(\boldsymbol{s},\boldsymbol{a})\big) < \epsilon, \tag{20}$$

where ϵ is equally set manually. Values for κ and ϵ are usually set close to 1.

For discrete environments, we can equally employ standard reinforcement learning techniques to obtain the Q-function $Q(\boldsymbol{s},\boldsymbol{a})$ and value function $V(\boldsymbol{s})$. In our experiments, we employed standard Q-learning [41] and LSPI [16] to obtain those quantities. In the discrete case, the temperature parameter η was set to 1.

3.2 Combining EM and Probabilistic Reinforcement Learning

As we have seen, the only difference between imitation learning and probabilistic reinforcement learning algorithms is the use of a weighted maximum likelihood (ML) estimate instead of a standard ML estimate. We can now combine the expectation maximization algorithm for discovering parametrized options with probabilistic reinforcement learning algorithms by weighting each time step in the maximization step of the EM algorithm.

Since the reinforcement learning weights $w_t = w(\boldsymbol{s}_t, \boldsymbol{a}_t)$ are independent of all latent variables, the derivations in Section 2 remain largely unaffected. The integration of the reinforcement learning weights affects only the maximization step. In the M-Step, all sample weights for the individual option components obtained by the EM algorithm are multiplied by the RL weights. Thus, we obtain the following combined weights of each sample

$$\begin{aligned} \tilde{w}_{B,j}(t) &= w_t \; p(o_t = j|\tau; \boldsymbol{\theta}^{\text{old}}), \\ \tilde{w}_O(t) &= w_t \; p(b_t = 1|\tau; \boldsymbol{\theta}^{\text{old}}), \\ \tilde{w}_{A,j}(t) &= w_t \; p(o_t = j|\tau; \boldsymbol{\theta}^{\text{old}}). \end{aligned}$$

Since the RL weightings only depend on the observed variables, we fortunately do not have to devise a new RL algorithm but can rely on a wide range of existing methods to provide the RL weights w_t. Here, the weights w_t could alternatively be obtained through other means, in non-RL settings. However, since we are interested in solving a reinforcement learning problem, we refer to them as RL weights. Thus, the proposed framework acts as an interface between existing RL algorithms and the policy updates. While traditional methods use the RL weights w_t to perform, for example, a maximum likelihood update of a monolithic policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, the proposed method estimates all elements of the option framework by fitting the hierarchical policy to the weighted state-action samples.

The information flow of the proposed algorithm is shown in Table 1.

4 Related Work

Options as temporally extended macro-actions were introduced by Sutton et al. [38]. While previous research leveraged the power of temporal abstraction [11, 28, 38], such efforts did not improve the sub-policies themselves. Improving the sub-policies based on the observed state-action-reward sequences is known as intra-option learning. Intra-option learning is a consequence of having Markov options and allows for updating all options that are consistent with an experience. While it is a desired property of option learning methods, it is not realized by all existing methods. Sutton et al. [37] showed that making use of intra-option learning can drastically improve the overall learning speed. Yet, the algorithms presented by Sutton et al. [37] relied on hand coded options and were presented in the discrete setting.

Options are also used in many hierarchical RL approaches, where they either extend the action space or are directly extended to sub-tasks, where the overall problem is broken up into potentially simpler sub-problems. Dietterich [8] proposed the MAXQ framework which uses several layers of such sub-tasks. However, the structure of these sub-tasks needs to be either specified by the user [8], or they rely on the availability of a successful trajectory [21]. Barto et al. [1] rely on artificial curiosity to define the reward signal of individual sub-tasks, where the agent aims to maximize its knowledge of the environment to solve new tasks quicker. This approach relies on salient events which effectively define the sub-tasks.

Stolle and Precup [35] first learn a flat solution to the task at hand and, subsequently, use state visitation statistics to build the option's initiation and termination sets. Mann and Mannor [18] apply the options framework to value iteration and show that it can speed up convergence.

Input: Number of sub-policies O, number of iterations L, number of episodes per iteration M, reinforcement learner $f_{\mathrm{RL}}(\boldsymbol{s}, \boldsymbol{a}, r)$.
Initialize $\pi(\boldsymbol{a}|\boldsymbol{s}, o; \boldsymbol{\theta}_A^{\mathrm{old}})$,$\pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O^{\mathrm{old}})$ and $\pi(b|\boldsymbol{s}, \bar{o}; \boldsymbol{\theta}_B^{\mathrm{old}})$.

for $l \leftarrow 1$ to L (# iterations)

Collect samples

for $i \leftarrow 1$ to M (# episodes)

Sample initial state $\boldsymbol{s}_{i,t}$ from environment.

Sample $b \sim \pi(b|\boldsymbol{s}, \bar{o}; \boldsymbol{\theta}_B^{\mathrm{old}})$

if $b = 1$

Sample option $o \sim \pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O^{\mathrm{old}})$,

Sample action $\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s}, o; \boldsymbol{\theta}_A^{\mathrm{old}})$.

Observe reward $r_{i,t}(\boldsymbol{a}_{i,t}, \boldsymbol{s}_{i,t})$ and next state $\boldsymbol{s}_{i,t+1}$.

Compute Sample Weights:

$\tilde{w}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow f_{\mathrm{RL}}(\boldsymbol{s}, \boldsymbol{a}, r)$.

Calculate Lower Bound:

Compute Messages:

$\alpha_t(o_t, b_t) \leftarrow p(\boldsymbol{a}_{1:t}, o_t, b_t|\boldsymbol{s}_{1:t}; \boldsymbol{\theta}^{\mathrm{old}})$

$\beta_t(o_t, b_t) \leftarrow p(\boldsymbol{a}_{t+1:T}, |\boldsymbol{s}_{t+1:T}, b_t, o_t; \boldsymbol{\theta}^{\mathrm{old}})$

Compute responsibilities:

$\gamma_t(o_t, b_t) \leftarrow \alpha_t(o_t, b_t)\beta_t(o_t, b_t)z_t^{-1}$

$\xi_t(o_{t:t+1}, b_{t:t+1}) \leftarrow \alpha_t(o_t, b_t)\beta_{t+1}(o_{t+1}, b_{t+1})$
$p(\boldsymbol{a}_{t+1}, o_{t+1}, b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}^{\mathrm{old}})z_t^{-1}$

Compute weights $w_{B,o}$, $w_{A,o}$, and w_O and target values $t_{B,o}$ and t_G .

Update Policy:

with weights, target values and state-action pairs.

Output: Policies $\pi^*(\boldsymbol{a}|\boldsymbol{s}, o)$, $\pi^*(o|\boldsymbol{s})$ and $\pi^*(b|\boldsymbol{s}, o)$.

Table 1: Learning options from experience. Termination events, options and actions are sampled from from the current policies. Subsequently, the distribution over latent variables is computed and weights f_{RL} are proposed by the RL algorithm The next policies are determined according to the update equations in the method section.

Option discovery approaches often aim to identify so called bottleneck states, i.e., states the agent has to pass pass through on its way from start to goal. McGovern and Barto proposed to formulate this intuition as a multiple-instance learning problem and solve it using a diverse density method [19]. Other approaches aim to find such bottleneck states using graph theoretic algorithms. The Q-Cut [22] and L-Cut [33] build transition graphs of the MDP and solve a min cut problem to find bottleneck states. Silver and Ciosek [32] assume a known MDP model to propose an option model composition framework, which can be used for planning while discovering new options. Niekum and Barto [25] present a method to cluster subgoals discovered by existing subgoal discovery methods to find skills that generalize across tasks. In the presented paper, we do not assume knowledge of the underlying MDP and, further, present a framework which is also suitable for the continuous setting.

In continuous state-action settings, several sub-task based approaches have been proposed. Ghavamzadeh and Mahadevan [10] proposed the use of a policy gradient method to learn subtasks while the selection of the sub-tasks is realized through Q-Learning. Morimoto and Doya [23] proposed to learn how to reach specific joint configurations of a robot as sub-tasks, such that these options can later be combined to learn more complicated tasks. In both approaches, the sub-

tasks have to be pre-specified by the user. Wingate et al. [42] use policy priors to encode desired effects like temporal correlation. Levy and Shimkin [17] propose to extend the state space by the option index, which allows for the use of policy gradient methods. In our proposed method, this option index is a latent variable which is inferred from data. The use of a latent variable allows our methods to update all options with all relevant data points. Konidaris and Barto [14] use the option framework to learn chaining of skills. This approach requires that the agent can reach the goal state before constructing the series of options leading to the goal state.

A concept similar to the options framework has been widely adapted in the field of robot learning. There, temporal abstraction is achieved through the use of movement primitives [27, 4]. Instead of learning policies over state-torque mappings to control robots, the agent learns parameters of a trajectory generator [13, 12]. Based on a Beta-Process Autoregressive HMM proposed by Fox et al. [9], Niekum et al. [26] proposed a method to segment demonstrated trajectories into a sequence of primitives, addressing the imitation learning problem. Ranchod et al. [31] extend the work of Fox et al. [9] to allow skill discovery in the inverse reinforcement learning setting. There, the task is to recover reward functions which lead to a skill based solution of a task. Compared to the proposed method, methods based on or similar to the Beta-Process Autoregressive HMM allow to extract skills or segments without a priori knowledge of the total number of skills. However, such methods are computationally expensive and have not been shown to work in the loop together with reinforcement learning methods.

Sequencing multiple primitives can be viewed as an approximation to the options framework for robot learning and allows more challenging tasks to be solved [36, 6]. Robot learning approaches often benefit from substantial task knowledge in the form of task demonstrations. Furthermore, a key benefit of these methods is the ability to offer a simple parametrization of complex behaviors, which also inspired the proposed approach. However, existing robot learning methods usually do not allow for intra-option learning from the complete trajectory data [5] and the individual primitives are generally of fixed duration [13].

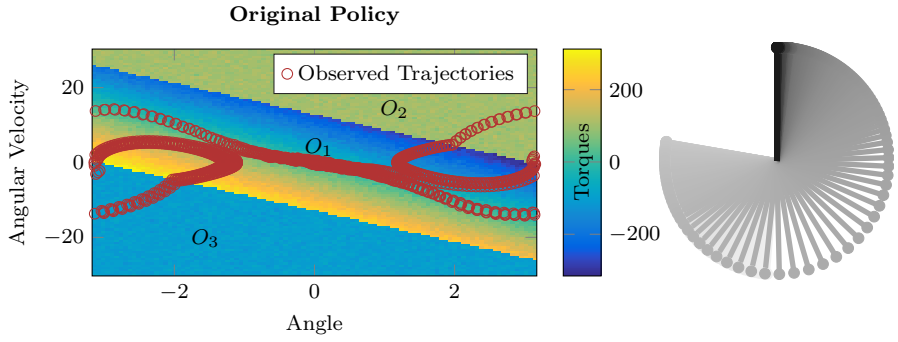
5 Evaluation

The evaluation of the proposed framework is separated in two parts. We first evaluated the imitation learning capabilities and, subsequently, proceeded to evaluate different reinforcement learning tasks as well as comparing the proposed methods to other option learning frameworks.

5.1 Imitation Learning

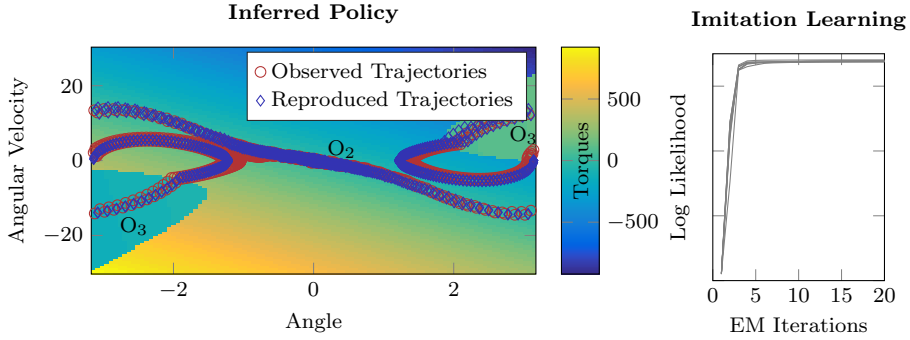
We started our evaluations with an imitation learning task. The evaluation of the imitation learning capabilities allowed us to ensure that the foundation of the proposed framework performs as expected.

For the imitation learning task, we evaluated the underpowered pendulum swing-up task. In this task, the agent exerted torques on the rotational joint of a pendulum and had to swing-up the pendulum into the upright position. However,



(a) NoisyHandCodedPolicy (b) Single Trajectory

Fig. 2: a) The hand coded policy consists of three bands, where the center band realizes a stabilizing controller and the outer bands accelerate the pendulum to enable a swing-up. b) One trajectory sample generated with the hand coded policy.



(a) Inferred Policy (b) LogLikelihood

Fig. 3: a) The policy learned through imitation learning. Red circles show the observations and blue diamonds show trajectories generated using this learned policy. The plot shows that the learned policy uses only two of three available options. Option 2 realizes the stabilizing policy and Option 3 realizes the accelerating policy. b) A qualitative plot showing the development of the log likelihood of the observed data under the learned policy for ten trials. The results show that convergence is usually reached after about five EM iterations.

the torques were limited such that the agent was not strong enough to perform a swing-up directly. Instead, the agent had to first perform a pre-swing to build up sufficient kinetic energy. The pendulum had a length of 0.5m, a mass of 10kg and a friction coefficient of 0.2Ns/m. The pendulum was internally simulated with a frequency of 10kHz. The agent controlled the pendulum at a frequency of 33Hz. The agent could exert at most 30Nm of torque and each episode had 100 time steps. This continuous task had two state variables, the angle and the velocity of the pendulum, where we used a periodic representation of the angle.

To generate observations, we provided a hand-coded policy which is shown in Fig. 2a. This policy was designed to generate noisy actions within $\pm$300Nm. However, the torques on the system were capped to the torque limit of 30Nm. The

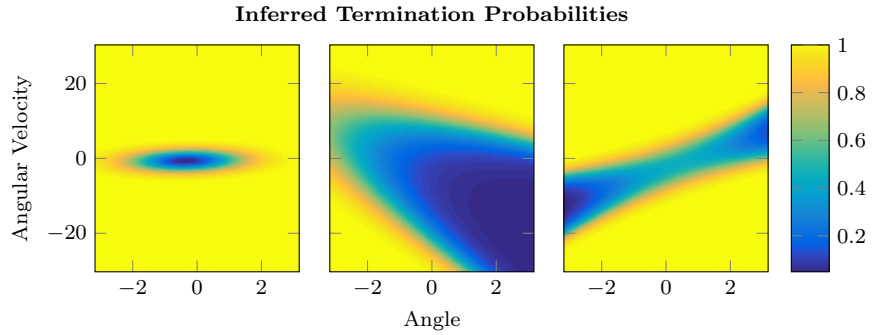


Fig. 4: Visualization of the Inferred Termination Policies for options one (left) through three (right). As shown in Fig. 3a, the first option is not used in the final policy and the agent learned to almost always terminate the first option. The third option learned a symmetric termination policy which terminates once the pendulum has sufficient energy to be 'pulled up' to the upright position. Interestingly, the second option learned a non-symmetric termination policy, possibly due to the relatively small number (five) of noisy demonstrations.

much larger range of desired torques was chosen to simplify the programming of the controller. This policy could successfully perform a pendulum swing-up if the pendulum was initially hanging down with zero velocity. An example trajectory generated by this hand coded policy is shown in Fig. 2b.

Based on five observed trajectories, we used the proposed framework to learn a policy with three options. However, the resulting policy, shown in Fig. 3a, learned to reproduce an effective policy using only two of the three available policies. Generally, we would not necessarily expect that the options recovered by the proposed algorithm exactly match the options of the demonstrator. The algorithm only aims at reproducing the observed behavior but is free to choose the internal structure of the hierarchical policy. Equally, in the RL case, we would not expect the proposed algorithm to learn options that are 'human-like'. Specifically, we would not expect that a robotic agent would use the same solution decomposition as a human operator.

Fig. 3a shows that the trajectories generated with this imitated policy closely resemble the observed trajectories. Fig. 4 shows the inferred termination policies of the different options. The results show that options will only be terminated once they are outside of their region of 'expertise'. Option one has a high termination probability in most regions, however, Fig. 3a shows that the final policy is actually not using this option. Fig. 3b shows the development of the log likelihood of the observed data under the imitated policy. The results show that convergence typically has been reached after about five iterations of the EM algorithm. In the imitation learning case, the learned solution is only valid if the system is initiated in a state which is similar to those states that were previously demonstrated. Outside of this region, the imitation learning solution will not be able to successfully solve the task and a reinforcement learning solution is required. In the proposed method, the activation policy will learn to initiate sub-policies according to their responsibility of state-space region. If, for example, the sub-policies are modeled as Gaussians and have infinite support, they have non-zero responsibility for all states and could, theoretically, be activated. If, however, a different class of proba-

bility distributions is more task-appropriate and has limited support, it would not be activated outside of its support region.

In this task, the activation policy was a soft-max distribution based on a squared expansion of the state features. The termination policies were represented by sigmoidal functions based on the same features as the activation policy. The sub-policies, however, were represented as linear Gaussian policies based directly on the state features.

5.2 Reinforcement Learning

We present results on three discrete state-action tasks as well as one continuous state-action task. In the discrete state action task settings, we compare to two existing option learning algorithms, namely the Q-Cut algorithm [22] as well as the L-Cut algorithm [34]. Both algorithms aim to find bottleneck states by solving a max flow/ min cut problem on the transition graph. After identifying such bottleneck states, options are generated which lead to these states. The main difference between these two algorithms is that Q-Cut aims to solve a global graph partitioning problem, while L-Cut aims to solve a local problem, based on transitions observed in one episode. The main difference to the proposed algorithm is that we do not aim to identify bottleneck states, but instead aim to automatically learn a decomposition of the problem that is suitable to the class of available option components.

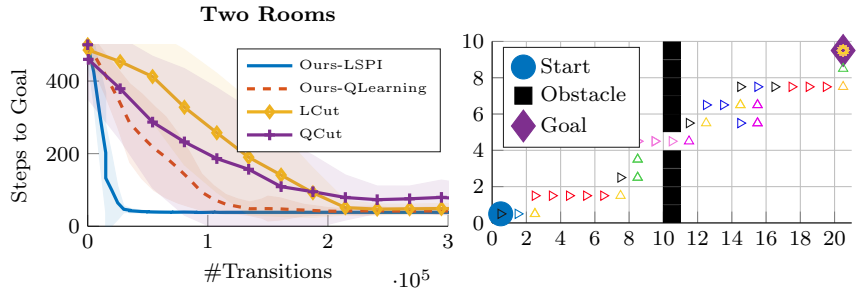
For all evaluations, we tested each setting ten times and report mean and standard deviation of the results.

5.2.1 Discrete Tasks.

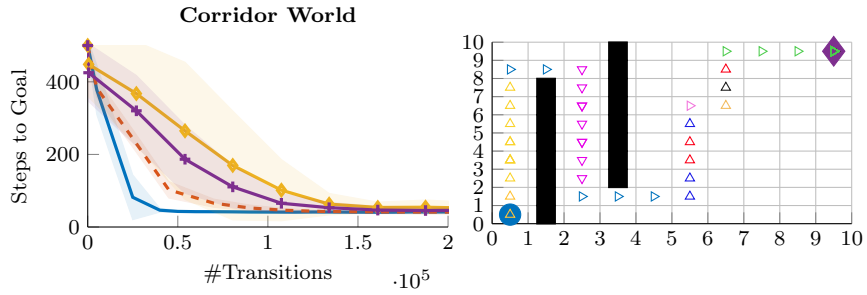
The discrete environments were given by three different gridworlds as shown in Figures 5b, 5d and 5f. The first world shown in Fig 5b represents the two rooms scenario [19], where the agent has to find a doorway to travel between two rooms. In the second world shown in Fig 5b, the agent has to traverse two elongated corridors before entering a big room in which the target is located. Finally, in the third world show in Fig. 5f no traditional bottleneck states appear, but the agent has to navigate around two obstacles. Furthermore, in this world optimal paths can lead around either side of the first obstacle.

In all experiments, the agent started in the lower level corner of the respective grid and had to traverse the grid while avoiding two obstacles to reach the goal at the opposite end. The actions available to the agent were going up, left, right and down. Transitions were successful with a probability of 0.8. Unsuccessful transitions had a uniform probability of ending up in any of the neighboring cells. The transition to each accessible field but the goal field generated a reward signal of -1. After reaching the goal, the agent received a reward of $+1$ for every remaining step of the episode, where each episode had a length of 500 time steps. If the agent tried to walk into an obstacle or to leave the field, it remained in the current position.

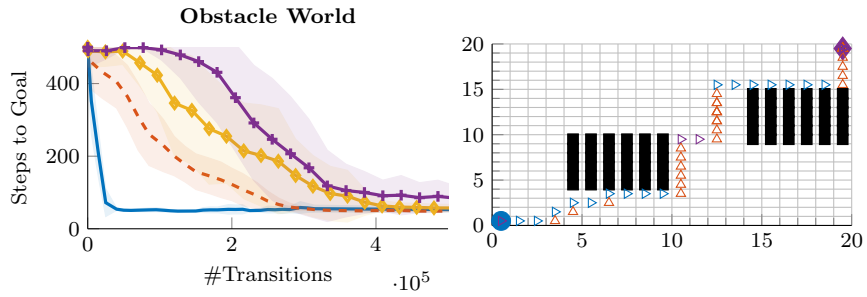
For the discrete tasks we used a tabular feature encoding for the flat policy. For the hierarchical policy, we used the tabular features for the activation and termi-



(a) Two Rooms Comparison (b) Two Rooms Setup

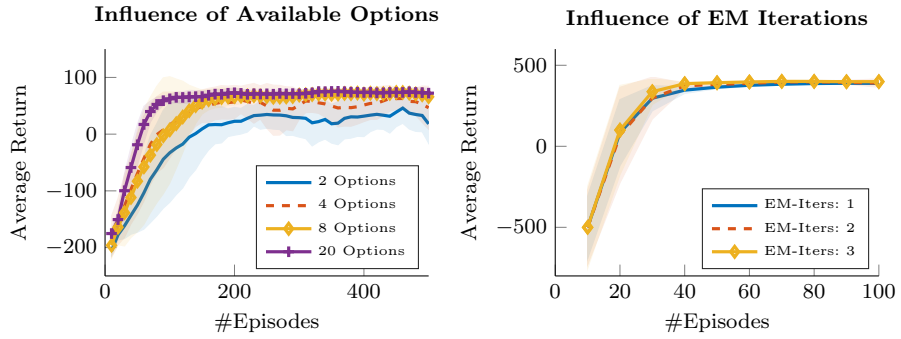


(c) Corridor World Comparison (d) Corridor World Setup



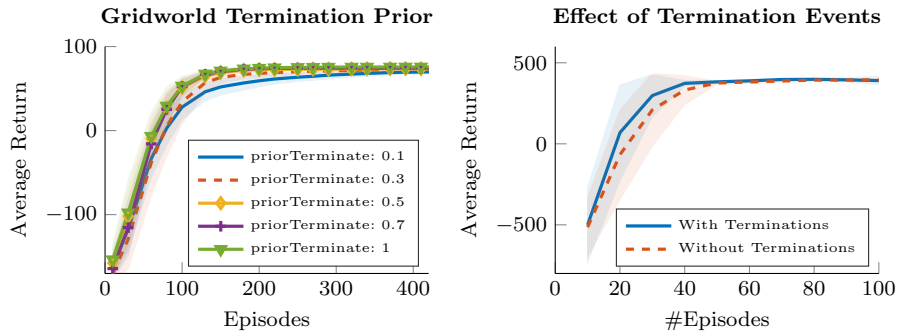
(e) Obstacle World Comparison (f) Obstacle World Setup

Fig. 5: b,d,f) Setups of the discrete world task. The agent has to cross fields of varying sizes while avoiding the obstacles on its way to the goal. Outlined triangles show trajectories learned using the proposed approach in combination with LSPI. The colors differentiate between active options. Since the transition dynamics were stochastic, the transitions did not always follow the selected actions. a,c,e) Learning performances of the different algorithms on the discrete world tasks. The results show that the L-Cut and Q-Cut generally had similar performance, where Q-Cut outperformed L-Cut in the first two worlds. However, in the relatively open obstacle world, L-Cut outperformed Q-Cut since Q-Cut was not able to find suitable bottleneck states. In all worlds the proposed approach outperformed both L-Cut and Q-Cut when using Q-Learning. Changing the RL algorithm to LSPI further increased the learning performance.



(a) Obstacle World #Options (b) Obstacle World #EM Iterations

Fig. 6: a) Evaluation of the proposed algorithm on a gridworld task. With an increased number of options the performance of the algorithm also increases. With eight options, optimal asymptotic performance is reached, however adding even more options further improves the learning speed. b)The number of EM iterations in the RL case does not have a large effect on the performance. More EM iterations improve the performance only marginally.



(a) Obstacle World Termination Prior (b) Obstacle World Terminations

Fig. 7: a) The effect of the termination prior (initialization of termination policies) using eight options. Changing the initial value of the termination probabilities within a reasonable range around 0.5 has only a small effect on the learning process. Very low values will lead to decreased performance. b) The effect of completely disabling terminations. Without terminations, the performance of the algorithm decreases.

nation policies. The sub-policies were state-independent multinomial distributions over the four actions.

In the discrete setting, we present results of two different RL learning methods, Q-Learning [41] and LSPI [16], in combination with the proposed framework and converted the resulting Q functions into RL weights as described in Section 3. However, especially when using Q-Learning, the resulting policy updates can be very different and may de-stabilize the learning process. To stabilize the learning process, we use a learning rate α for our policy updates such that

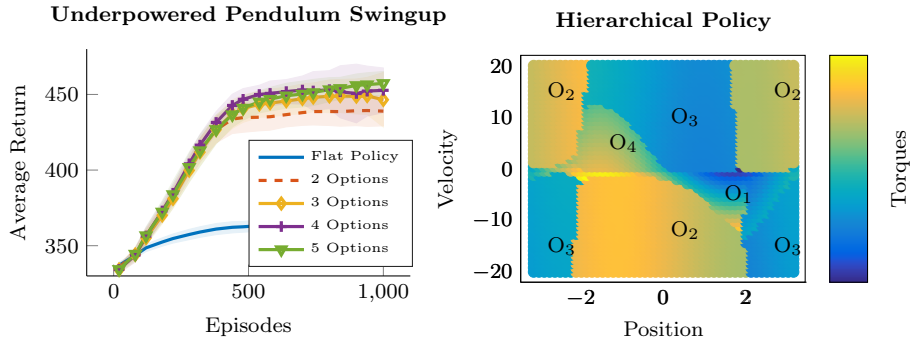
$$\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s},\bar{o}) = \alpha\pi(\boldsymbol{a}|\boldsymbol{s},\bar{o}) + (1-\alpha)\hat{\pi}(\boldsymbol{a}|\boldsymbol{s},\bar{o}),$$

where $\pi(\boldsymbol{a}|\boldsymbol{s}, \bar{o})$ is the resulting policy of the EM update and $\hat{\pi}(\boldsymbol{a}|\boldsymbol{s}, \bar{o})$ is the previous policy. Alpha was set to 0.1 in the experiments. Alternatively, the learning rate of Q-Learning itself could be decreased. However the proposed scheme yields better performance.

Comparison To Existing Methods. In the comparative results to related work we follow the therein established method of reporting 'steps to goal' as qualitative measure. In the remaining evaluations of our algorithm we report the average return, which may in some cases be more informative since our reward functions also punish 'falling off' the board. The results in Fig. 5 show that in all experiments the proposed framework learned solutions faster than both the Q-Cut as well as the L-Cut methods. Comparing the use of Q-Learning and LSPI in the proposed framework, the results show that LSPI leads to convergence considerably faster than Q-Learning. Since the structure of the individual action policies learned by the proposed approach was given simply as a distribution over the four possible actions, the converged sub-policies usually always select only one action. This simplicity of the sub-policies is a key factor to accelerate the overall learning speed. While we do not present results of comparisons to primitive-based methods such as, for example, using Q-Learning directly, both of the methods that we did compare to have shown to outperform Q-Learning. Thus, we compared to such primitive-based methods indirectly. In our experience, both Q-Cut and L-Cut outperform Q-Learning when not using experience replay. However, in our internal evaluations on the tasks presented in this paper, Q-Learning with experience replay resulted in performance levels similar to Q-Cut and L-Cut, but worse than the proposed method.

Influence of Available Options. After comparing to existing methods, we further evaluated the properties of the proposed framework. All remaining evaluations were performed using LSPI in the obstacle world. In our experience, these results were representative of both using Q-Learning as well as performing them in different tasks. Fig. 6a shows the influence of available options. In theory this task can be solved optimally with only two options, where one will always go right and one will always go up. However, the results show that making more options available to the algorithm improved both asymptotic performance as well as speed of convergence. Adding more than 20 options did not further increase the performance.

Influence of EM Iterations. The proposed EM style of computing policy updates can be costly and, generally, requires several iterations as also seen in the imitation learning case in Fig. 3b. However, in the RL setting, subsequent policies are expected to be similar and, thus, a small number of EM iterations should be sufficient. The results in Fig. 6b show that this intuition holds true in our evaluations. In our experience, performing even more EM iterations did not change the result. However, when using very few rollouts per iteration, the effect of performing multiple EM iterations can be more pronounced. However, even when a small number of EM iterations is sufficient, introducing these additional computations results in a noticeable computational overhead in continuous task. Here, especially the number of options is an important factor influencing the runtime of the EM algorithm. In our experience, the addition of the EM algorithm would lead to approximately twice the overall computational requirements. However, this factor is



(a) Pendulum Number of Options (b) Hierarchical Policy

Fig. 8: (a) Evaluation of the proposed algorithm on the underpowered pendulum swingup task. A flat linear policy is insufficient to solve the task, only the combination of multiple linear policies is sufficiently expressive. Using two or three options, the task can be solved but stabilization depends on bang-bang control. With four or more options, the policy is sufficiently expressive to incorporate high torque signals for the swing-up and finely tuned sub-policies for stabilization. (b) Visualization of the hierarchical policy composed of four options. Options 2 and 3 learned a bang-bang control scheme and options 1 and 4 learned to stabilize the pendulum around the upright position. (Best viewed in color).

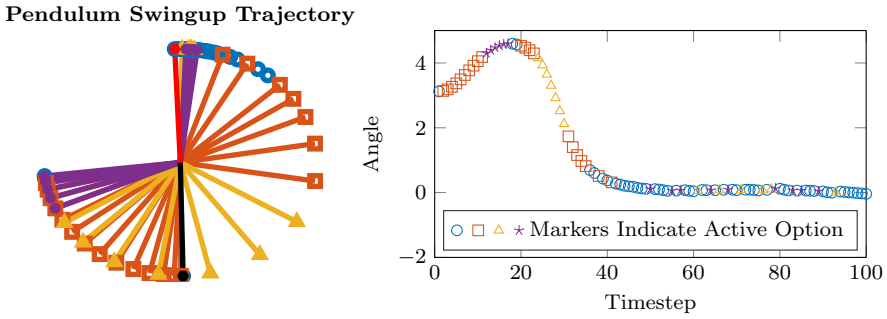
of course highly dependent on the RL method used. In the discrete tasks that we evaluated, the computational overhead due to the EM iterations was negligible.

Influence of Termination Events. Finally, we evaluated the influence of the probabilistic terminations. In the proposed framework, the sub-policies have to be initialized and, thus, a prior for the termination policies has to be set. Fig. 7a shows the effect of changing this prior. The results show that the proposed framework is robust to wide range of these initializations.

We also evaluated the effect of disabling the probabilistic termination sub-policies. In this case, the algorithm could still learn multiple options but no termination policies. Thus, each option could not be active for more than one time step but terminated after every step. The results in Fig. 7b show that learning without terminations slowed down the convergence speed. In our experience, this effect was strongly linked to the stochasticity of the transitions. The higher this stochasticity was, the stronger the benefit of the termination policies became.

5.2.2 Continuous Task.

After evaluating our algorithm on several discrete tasks, we returned to the continuous pendulum-swingup task described in the imitation learning section of the results. As before, the task is to swing-up an underpowered pendulum. To solve this task the agent has to learn to first perform a pre-swing to build up kinetic energy and then use the momentum to fulfill the task. Furthermore, in the presented task two solutions are possible, performing the swing-up clock-wise or counter-clock-wise. To provide a reinforcement learning signal for the continuous task, we employed the Hierarchical Relative Entropy Policy Search (HiREPS) algorithm [5],



(a) Trajectory (b) Trajectory in state space

Fig. 9: (a) Visualization of swingup trajectory color coded by the currently active option. The pendulum is initially hanging down with a small rotation. First option 2 performs a pre-swing and, subsequently, option 3 accelerates the pendulum until options 1 and 4 start the stabilization process. Around timestep 15, option 4 is active for a short time, but the kinetic energy at that point is insufficient for a direct swingup. (Best viewed in color).

where we used Fourier basis features as described by Konidaris et al. [15] to represent the value function. For the value function a feature expansion of the fifth order was used. For the activation policy as well as the termination policies, a squared feature expansion was used. The individual sub-policies worked directly on the state observations, using only an additional constant offset. Thus, the sub-policies were linear controllers. In the pendulum swing-up task, a single linear controller is insufficient and, thus, the proposed approach had to combine multiple sub-policies to achieve the necessary non-linear behavior required for a swing-up.

The results in Figure 8a show that while a single linear policy was insufficient to solve this task, it could be solved using two options. Adding more options further improved the resulting hierarchical policy. The visualization of the resulting policy in Figure 8b shows that with more options, the algorithm learned a control scheme where options two and three were used to swing up the pendulum, and options one and four incorporated a linear stabilization scheme around the upright position of the pendulum. Figure 9a shows a trajectory generated by the resulting policy. Starting from the bottom, the pendulum was first accelerated by options two and three. The plot shows that in-between options two and three, option four was active for a few time steps. However, the kinetic energy at that point was insufficient to fully swing up the pendulum. After the pendulum almost reached the upright position around time step 40, the stabilizing options took over control. Since all option components are stochastic distributions, some option switches still occur even after the pendulum is stabilized. Since the effect of switching into a different option in the stable position for a single time step could easily be balanced by activating the stabilizing option in the next time step, the agent did not have a strong incentive to learn to avoid such behavior. Letting the algorithm run for more iterations might further improve this behavior.

5.2.3 Limitations.

While the experiments show that the presented method worked well in the scenarios that were evaluated, we also want to make explicit the assumptions made in

this paper. Primarily, the proposed method expects that the number of required options is known a priori. In our experience, this requirement is rather benign in practice, as the algorithm can be initialized with an excessive amount without deterioration in quality of the solution. However, adding more options does increase the computational requirements. Thus, approaches for automatically generating a task-appropriate number of options is an important aspect of future work. Furthermore, we introduced a damping factor $\alpha = 0.1$ on the policy update for the discrete setting, which we found to be especially important when using Q-Learning as the underlying RL method. In our experience, the recommended value of α depends on the RL method used as well as the task under consideration. Methods such as LSPI will generally work well with larger values of α.

6 Conclusion & Future Work

In this paper, we presented a method to estimate the components of the option framework from data. The results show that the proposed method is able to learn options in the discrete and continuous setting. In the discrete setting, the algorithm performs better than two related option-discovery algorithms which are based on exploiting bottle-neck states. Instead of relying on bottle-neck states, the proposed algorithm achieves its performance by combining options with simpler sub-policies.

In the continuous setting, the results show that the algorithm is able to solve a non-linear task using a combination of options with only linear sub-policies. In this setting, a single linear policy is insufficient for solving the task. Furthermore, the framework allows for parametrized policies and, thus, state-of-the-art policy search methods developed for flat policies can be used to learn hierarchical policies.

The presented approach infers the option's structure, such as the activation policy and termination policies, from data. However, the number of options still has to be set a-priori by the practitioner. While the results show that setting a relatively large number of options typically yields good performance, learning the number of options is an important aspect of future work. Finally, while the presented framework estimates the most likely termination policies, finding a way of enforcing fewer terminations might further improve learning performance.

References

1. 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.
2. L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.
3. Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, 2006.
4. B. Da Silva, G. Konidaris, and A.G. Barto. Learning parameterized skills. *In Proceedings of the International Conference on Machine Learning (ICML)*, 2012.
5. C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.
6. 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.
7. P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–271. Morgan Kaufmann Publishers, 1993.
8. T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.
9. E. B. Fox, M. I. Jordan, E. B. Sudderth, and A. S. Willsky. Sharing features among dynamical systems with beta processes. In *Advances in Neural Information Processing Systems (NIPS)*, pages 549–557, 2009.
10. M. Ghavamzadeh and S. Mahadevan. Hierarchical Policy Gradient Algorithms. In *Proceedings of the International Conference for Machine Learning (ICML)*, 2003.
11. L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1993.
12. S. Kajita, K. Kanehiro, F. andi Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped Walking Pattern Generation by using Preview Control of Zero-Moment Point. In *Proceedings of the IEEE International Conference of Robotics and Automation (ICRA)*, 2003.
13. J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, pages 1–33, 2010.
14. G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.
15. G. Konidaris, S. Osentoski, and P.s. Thomas. Value function approximation in reinforcement learning using the fourier basis. *Conference on Artificial Intelligence (AAAI)*, 2011.
16. M. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, December 2003.
17. 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.
18. T. A. Mann and S. Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.
19. A. McGovern and A. G. Barto. International conference on machine learning (icml). *Computer Science Department Faculty Publication Series*, page 8, 2001.
20. A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *International Conference on Machine Learning (ICML)*, page 8, 2001.
21. N. Mehta, S. Ray, P. Tadepalli, and T. G. Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.
22. I. Menache, S. Mannor, and N. Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2002.
23. 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.

24. Andrew Ng and Adam Coates. Autonomous Inverted Helicopter Flight via Reinforcement Learning. *Experimental Robotics IX*, 1998.
25. S. Niekum and A. G. Barto. Clustering via dirichlet process mixture models for portable skill discovery. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.
26. S. Niekum, S. Osentoski, G.D. Konidaris, and A.G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
27. A. Paraschos, C. Daniel, J. Peters, and G Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.
28. R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems (NIPS)*, 1998.
29. J. Peters, K. Mülling, and Y. Altun. Relative Entropy Policy Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2010.
30. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.
31. P. Ranchod, B. Rosman, and G. Konidaris. Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In *Intelligent Robots and Systems (IROS), 2015*, pages 471–477. IEEE, 2015.
32. D. Silver and K. Ciosek. Compositional Planning Using Optimal Option Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.
33. Ö. Simsek and A. G. Barto. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.
34. Ö. Simsek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.
35. M. Stolle and D. Precup. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation*, pages 212–223. Springer, New York City, 2002.
36. F. Stulp and S. Schaal. Hierarchical Reinforcement Learning with Movement Primitives. In *Proceedings of the IEEE International Conference on Humanoid Robots (HUMANOIDS)*, 2012.
37. 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.
38. 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, 1999.
39. 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.
40. 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.
41. Christopher J. C. H. Watkins and Peter Dayan. Q-Learning. *Machine Learning*, 8(3-4): 279–292, 1992.
42. 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.