The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Hierarchical Policy Search via Return-Weighted Density Estimation

Takayuki Osa
University of Tokyo
277-0882, Chiba, Japan
RIKEN Center for AIP
103-0027, Tokyo, Japan

Masashi Sugiyama
RIKEN Center for AIP
103-0027, Tokyo, Japan
University of Tokyo
277-0882, Chiba, Japan

Abstract

Learning an optimal policy from a multi-modal reward function is a challenging problem in reinforcement learning (RL). *Hierarchical* RL (HRL) tackles this problem by learning a hierarchical policy, where multiple option policies are in charge of different strategies corresponding to modes of a reward function and a gating policy selects the best option for a given context. Although HRL has been demonstrated to be promising, current state-of-the-art methods cannot still perform well in complex real-world problems due to the difficulty of identifying modes of the reward function. In this paper, we propose a novel method called *hierarchical policy search via return-weighted density estimation* (HPSDE), which can efficiently identify the modes through density estimation with return-weighted importance sampling. Our proposed method finds option policies corresponding to the modes of the return function and automatically determines the number and the location of option policies, which significantly reduces the burden of hyper-parameters tuning. Through experiments, we demonstrate that the proposed HPSDE successfully learns option policies corresponding to modes of the return function and that it can be successfully applied to a motion planning problem of a redundant robotic manipulator.

Introduction

Recent work on reinforcement learning (RL) has been successful in various tasks, including robotic manipulation (Gu et al. 2017; Levine et al. 2016b; 2016a) and playing a board game (Silver et al. 2016). However, many RL methods cannot leverage a hierarchical task structure, whereas many tasks in the real world are highly structured. Grasping is a good example of such structured tasks. When grasping an object, humans know multiple grasp types from their experience and adaptively decide how to grasp the given object (Cutkosky and Howe 1990; Napier 1956). This strategy can be interpreted as a hierarchical policy where the gating policy first selects the grasp type and the option policy that represents the selected grasp type subsequently plans the grasping motion (Osa, Peters, and Neumann 2016). Prior work on hierarchical RL suggests that learning various option policies increases the versatility (Daniel et al. 2016) and that exploiting a hierarchical task structure can exponentially reduce the search space (Dietterich 2000).

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

However, RL of a hierarchical policy is not a trivial problem. As indicated by Daniel et al. (2016), each option policy in a hierarchical policy needs to focus on a single mode of the return function, otherwise a learned policy will try to average multiple modes and fall into a local optimum with poor performance. Therefore, it is necessary to properly assign option policies to individual modes of the return function, and the challenge of this problem is how to identify the number and locations of modes of the return function. Although regularizers (Bacon, Harb, and Precup 2017; Florensa, Duan, and Abbeel 2017) can be used to drive the option policies to various solutions, they cannot prevent an option policy from averaging over multiple modes of the return function. Additionally, in existing methods, the performance often depends on initialization or pre-training of the policy (Daniel et al. 2016; Florensa, Duan, and Abbeel 2017), and a user often needs to specify the number of the option policies in advance, which significantly affects the performance (Bacon, Harb, and Precup 2017).

To address such issues in existing hierarchical RL, we propose model-free hierarchical policy search via return-weighted density estimation (HPSDE). Our approach reduces the problem of identifying the modes of the return function to estimating the return-weighted sample density. Unlike previous methods, the number and the location of the option policies is automatically determined without explicitly estimating the option policy parameters, and an option policy learned by HPSDE focuses on a single mode of the return function. We discuss the connection between the expected return maximization and the density estimation with return-weighted importance. The experimental results show that HPSDE outperforms a state-of-the-art hierarchical RL method and that HPSDE finds multiple solutions in motion planning for a robotic redundant manipulator.

Problem Formulation

We consider a reinforcement learning problem in the Markov decision process (MDP) framework, where an agent is in a state $\boldsymbol{x} \in \mathcal{X}$ and takes an action $\boldsymbol{u} \in \mathcal{U}$. In this paper, we concentrate on the episodic case of hierarchical RL, where a policy $\pi(\boldsymbol{\tau}|\boldsymbol{x}_0)$ generates a trajectory $\boldsymbol{\tau}$ for a given initial state $\boldsymbol{x}_0$ and only one selected policy is executed until the end of the episode. After every episode, the agent receives a return of a trajectory $R(\boldsymbol{\tau}, \boldsymbol{x}_0) = \sum_{t=0}^{T} r_t$, which

is given by a sum of immediate rewards. T is the length of the trajectory, which is a random variable. In the following, we denote by $\boldsymbol{s} = \boldsymbol{x}_0$ the initial state of the episode, which is often referred to as the "context", and we assume that a trajectory $\boldsymbol{\tau}$ contains all the information of actions $\boldsymbol{u}_t$ and the state $\boldsymbol{x}_t$ during the episode. The purpose of policy search is to obtain the policy $\pi(\boldsymbol{\tau}|\boldsymbol{s})$ that maximizes the expected return (Deisenroth, Neumann, and Peters 2013)

$$J(\pi) = \iint d(\boldsymbol{s})\pi(\boldsymbol{\tau}|\boldsymbol{s})R(\boldsymbol{s},\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}, \tag{1}$$

where $d(\boldsymbol{s})$ is the distribution of the context $\boldsymbol{s}$. In hierarchical RL, we consider a policy that is given by a mixture of option policies:

$$\pi(\boldsymbol{\tau}|\boldsymbol{s}) = \sum_{o\in\mathcal{O}} \pi(o|\boldsymbol{s})\pi(\boldsymbol{\tau}|\boldsymbol{s},o), \tag{2}$$

where o is a latent variable which represents the option of the policy and $\mathcal{O}$ is a set of the latent variable o in a hierarchical policy. We refer to $\pi(\boldsymbol{\tau}|\boldsymbol{s},o)$ as an option policy and $\pi(o|\boldsymbol{s})$ as a gating policy. Here, the option is defined over trajectories, unlike the option over actions as in Machado, Bellemare, and Bowling (2017); Bacon, Harb, and Precup (2017). The expected return of a hierarchical policy is given by

$$J(\pi) = \sum_{o\in\mathcal{O}} \iint d(\boldsymbol{s})\pi(o|\boldsymbol{s})\pi(\boldsymbol{\tau}|o,\boldsymbol{s})R(\boldsymbol{s},\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}. \tag{3}$$

The goal of hierarchical policy search is to learn an optimal hierarchical policy π^* that maximizes the expected return $J(\pi)$:

$$\pi^*(\boldsymbol{\tau}|\boldsymbol{s}) = \sum_{o\in\mathcal{O}} \pi^*(o|\boldsymbol{s})\pi^*(\boldsymbol{\tau}|\boldsymbol{s},o) = \arg\max_{\pi} J(\pi). \tag{4}$$

This problem can be solved with respect to samples $\mathcal{D} = \{(\boldsymbol{s}_i,\boldsymbol{\tau}_i,R_i(\boldsymbol{s}_i,\boldsymbol{\tau}_i))\}$ collected through rollouts. The challenge of hierarchical policy search is to estimate the latent variable o, which we cannot observe. The expected return for the policy π in (3) can be rewritten as

$$J(\pi) = \sum_{o\in\mathcal{O}} p(o)\mathbb{E}[R|\pi(\boldsymbol{\tau}|\boldsymbol{s},o),o], \tag{5}$$

where the expectation $\mathbb{E}[R|\pi(\boldsymbol{\tau}|\boldsymbol{s},o),o]$ is given by

$$\mathbb{E}[R|\pi(\boldsymbol{\tau}|\boldsymbol{s},o),o] = \iint d(\boldsymbol{s}|o)\pi(\boldsymbol{\tau}|\boldsymbol{s},o)R(\boldsymbol{s},\boldsymbol{\tau})\mathrm{d}\boldsymbol{s}\mathrm{d}\boldsymbol{\tau}, \tag{6}$$

and $d(\boldsymbol{s}|o)$ is the distribution of contexts $\boldsymbol{s}$ assigned to the option o. When the assignment of each sample $(\boldsymbol{s}_i,\boldsymbol{\tau}_i,R(\boldsymbol{s}_i,\boldsymbol{\tau}_i))$ to the option o is known, maximizing $\mathbb{E}[R|\pi(\boldsymbol{\tau}|\boldsymbol{s},o),o]$ is equivalent to solving a policy search problem with respect to samples $\{(\boldsymbol{s}_i^o,\boldsymbol{\tau}_i^o,R(\boldsymbol{s}_i^o,\boldsymbol{\tau}_i^o)))\}$ assigned to the option o. Therefore, the formulation in (5) indicates that, if we can estimate the latent variable o_i for each sample $(\boldsymbol{s}_i,\boldsymbol{\tau}_i)$, the option policy $\pi(\boldsymbol{\tau}|\boldsymbol{s},o)$ can be learned using a policy search method for a monolithic policy, e.g., trust region policy optimization (TRPO) (Schulman et al. 2015) or relative entropy policy search (REPS) (Peters, Mülling, and Altun 2010).

As discussed above, estimation of the latent variable o is a crucial problem in hierarchical RL. Next, we introduce the return-weighted density estimation to address this problem.

Hierarchical Policy Search via Return-Weighted Density Estimation

In this section, we introduce our proposed method, hierarchical policy search via return-weighted density estimation (HPSDE). To learn the number and the location of option policies, we estimate the latent variable o through estimating the trajectory density induced by the unknown optimal policy π^*. This density estimation is performed by using the return-weighted importance sampling. In the following, we discuss the details of 1) the return-weighted importance weight, 2) the latent variable estimation, and 3) the gating policy for selecting the option policies.

Return-Weighted Density Estimation

To estimate the latent variable o, we consider the trajectory density induced by the optimal policy, $p^*(\boldsymbol{s},\boldsymbol{\tau}) = d(\boldsymbol{s})\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$. Here, we assume that the optimal policy draws samples that lead to higher returns with higher probability. This assumption is equivalent to assuming that the optimal policy $\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$ is of the form

$$\pi^*(\boldsymbol{\tau}|\boldsymbol{s}) = \frac{f\left(R(\boldsymbol{s},\boldsymbol{\tau})\right)}{Z}, \tag{7}$$

where Z is a partition function, and $f(R)$ is a *functional*, which is a function of the return function. $f(R)$ should be monotonically increasing with respect to R such that a trajectory with a higher return is generated with higher probability by the optimal policy. This assumption is commonly used in prior work on policy search (Deisenroth, Neumann, and Peters 2013). Under this assumption, finding the modes of the return function $R(\boldsymbol{s},\boldsymbol{\tau})$ is equivalent to finding the modes of the density $p^*(\boldsymbol{s},\boldsymbol{\tau})$ induced by the optimal policy $\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$.

Since the optimal policy $\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$ is unknown, we cannot directly sample $\{(\boldsymbol{s}_i^*,\boldsymbol{\tau}_i^*)\} \sim d(\boldsymbol{s})\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$. Thus, we use an importance sampling technique to estimate the density $d(\boldsymbol{s})\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$ induced by the optimal policy $\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$.

We collect samples $\{(\boldsymbol{s}_i,\boldsymbol{\tau}_i,R(\boldsymbol{s}_i,\boldsymbol{\tau}_i))\}_{i=1}^N$ drawn from the current policy $\pi_{\text{old}}(\boldsymbol{\tau}|\boldsymbol{s})$ and a given context distribution $d(\boldsymbol{s})$. The importance of the sample $(\boldsymbol{s}_i,\boldsymbol{\tau}_i)$ can be given by

$$W(\boldsymbol{s}_i,\boldsymbol{\tau}_i) = \frac{d(\boldsymbol{s}_i)\pi^*(\boldsymbol{\tau}_i|\boldsymbol{s}_i)}{d(\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{\tau}_i|\boldsymbol{s}_i)} = \frac{\pi^*(\boldsymbol{\tau}_i|\boldsymbol{s}_i)}{\pi_{\text{old}}(\boldsymbol{\tau}_i|\boldsymbol{s}_i)} \tag{8}$$

$$= \frac{f\left(R(\boldsymbol{s}_i,\boldsymbol{\tau}_i)\right)}{Z\pi_{\text{old}}(\boldsymbol{\tau}_i|\boldsymbol{s}_i)}. \tag{9}$$

Note that the context distribution $d(\boldsymbol{s})$ is given by an environment and invariant to the policy in an episodic case. If we normalize W, we obtain the normalized weight

$$\tilde{W}(\boldsymbol{s}_i,\boldsymbol{\tau}_i) = \frac{W(\boldsymbol{s}_i,\boldsymbol{\tau}_i)}{\sum_{j=1}^N W(\boldsymbol{s}_j,\boldsymbol{\tau}_j)} \tag{10}$$

$$= \frac{\frac{f(R(\boldsymbol{s}_i,\boldsymbol{\tau}_i))}{Z\pi_{\text{old}}(\boldsymbol{\tau}_i|\boldsymbol{s}_i)}}{\sum_{j=1}^N \frac{f(R(\boldsymbol{s}_j,\boldsymbol{\tau}_j))}{Z\pi_{\text{old}}(\boldsymbol{\tau}_j|\boldsymbol{s}_j)}} = \frac{\frac{f(R(\boldsymbol{s}_i,\boldsymbol{\tau}_i))}{\pi_{\text{old}}(\boldsymbol{\tau}_i|\boldsymbol{s}_i)}}{\sum_{j=1}^N \frac{f(R(\boldsymbol{s}_j,\boldsymbol{\tau}_j))}{\pi_{\text{old}}(\boldsymbol{\tau}_j|\boldsymbol{s}_j)}}. \tag{11}$$

Therefore, we can compute the importance $\tilde{W}(\boldsymbol{s}_i,\boldsymbol{\tau}_i)$ of each sample even though the partition function Z of the

optimal policy π^* is unknown. Thus, we can estimate the density induced by the optimal policy $d(\boldsymbol{s})\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$ by using $\{(\boldsymbol{s}_i, \boldsymbol{\tau}_i, R(\boldsymbol{s}_i, \boldsymbol{\tau}_i))\}_{i=1}^N \sim d(\boldsymbol{s})\pi_{\text{old}}(\boldsymbol{\tau}|\boldsymbol{s})$ with the importance weight $\tilde{W}$. We refer to the density estimation using the importance weight $\tilde{W}$ as *return-weighted density estimation*.

Learning the optimal policy can be formulated as the distribution matching between the trajectory densities induced by the optimal policy and the learner's policy

$$\hat{\pi}(\boldsymbol{\tau}|\boldsymbol{s}) = \arg\min_{\pi} D_{\text{KL}}(p^*(\boldsymbol{s}, \boldsymbol{\tau})||p_\pi(\boldsymbol{s}, \boldsymbol{\tau})), \tag{12}$$

where $D_{\text{KL}}(p^*(\boldsymbol{s}, \boldsymbol{\tau})||p_\pi(\boldsymbol{s}, \boldsymbol{\tau}))$ is the KL divergence given by

$$D_{\text{KL}}(p^*(\boldsymbol{s}, \boldsymbol{\tau})||p_\pi(\boldsymbol{s}, \boldsymbol{\tau})) = \iint p^*(\boldsymbol{s}, \boldsymbol{\tau}) \log \frac{p^*(\boldsymbol{s}, \boldsymbol{\tau})}{p_\pi(\boldsymbol{s}, \boldsymbol{\tau})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$= \iint W(\boldsymbol{s}, \boldsymbol{\tau}) p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau}) \log \frac{W(\boldsymbol{s}, \boldsymbol{\tau}) p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau})}{p_\pi(\boldsymbol{s}, \boldsymbol{\tau})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}, \tag{13}$$

$p^*(\boldsymbol{s}, \boldsymbol{\tau})$ is the density induced by the optimal policy, $p_\pi(\boldsymbol{s}, \boldsymbol{\tau})$ is the density induced by the newly learned policy, and $p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau})$ is the density induced by the old policy used for data collection. The minimizer of $D_{\text{KL}}(p^*(\boldsymbol{s}, \boldsymbol{\tau})||p_\pi(\boldsymbol{s}, \boldsymbol{\tau}))$ is given by the maximizer of the weighted log likelihood:

$$L(\pi, \pi_{\text{old}}) = \iint W(\boldsymbol{s}, \boldsymbol{\tau}) p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau}) \log p_\pi(\boldsymbol{s}, \boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$\approx \frac{1}{N} \sum_{(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \in \mathcal{D}} \tilde{W}(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \log p_\pi(\boldsymbol{s}_i, \boldsymbol{\tau}_i). \tag{14}$$

The connection to maximizing the expected return can be seen as follows. If we assume that $R > 0$ and the optimal policy follows $\pi^*(\boldsymbol{\tau}|\boldsymbol{s}) = R(\boldsymbol{s}, \boldsymbol{\tau})/Z$, in a manner similar to the results shown by Dayan and Hinton (1997) and Kober and Peters (2011), we can obtain

$$\log J(\pi) = \log \iint p_\pi(\boldsymbol{s}, \boldsymbol{\tau}) R(\boldsymbol{\tau}, \boldsymbol{s}) \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$= \log \iint p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau}) \frac{d(\boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s})}{p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau})} \frac{p_\pi(\boldsymbol{s}, \boldsymbol{\tau})}{d(\boldsymbol{s})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$\geq \iint p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau}) \frac{d(\boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s})}{p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau})} \log \frac{p_\pi(\boldsymbol{s}, \boldsymbol{\tau})}{d(\boldsymbol{s})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$= Z \iint p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau}) \frac{R(\boldsymbol{\tau}, \boldsymbol{s})}{\pi_{\text{old}}(\boldsymbol{\tau}|\boldsymbol{s}) Z} \log \frac{p_\pi(\boldsymbol{s}, \boldsymbol{\tau})}{d(\boldsymbol{s})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$\propto L(\pi, \pi_{\text{old}}) + \text{const.} \tag{15}$$

Therefore, maximizing the weighted log likelihood $L(\pi, \pi_{\text{old}})$ is equivalent to maximizing the lower bound of the expected return in this case.

In practice, the return function R and $f(\cdot)$ in (7) need to be designed by a practitioner. To argue the optimal form of $f(\cdot)$ in (7), we need to make further assumptions. For example, the optimal policy of the form $\pi^*(\boldsymbol{\tau}|\boldsymbol{s}) = \exp\left(R(\boldsymbol{s}, \boldsymbol{\tau})\right)/Z$ can be justified based on the maximum entropy principle, which is often used in the literature of inverse RL (Ziebart 2010). In this work, we leave the form of $f(\cdot)$ in (7) as an open design choice for a practitioner.

Latent Variable Estimation

Based on the discussion in the previous section, we estimate the latent variable o of a hierarchical policy by matching the trajectory densities induced by the optimal policy and the learner's policy. Specifically, we consider the marginal likelihood:

$$p(\boldsymbol{s}, \boldsymbol{\tau}) = \sum_{o \in \mathcal{O}} p(o) p(\boldsymbol{s}, \boldsymbol{\tau}|o) \tag{16}$$

and estimate the latent variable o by minimizing

$$D_{\text{KL}}\left(p^*(\boldsymbol{s}, \boldsymbol{\tau}) \middle\| \sum_{o \in \mathcal{O}} p(o) p(\boldsymbol{s}, \boldsymbol{\tau}|o)\right), \tag{17}$$

which can be solved by maximizing the weighted log marginal likelihood

$$\iint W(\boldsymbol{s}, \boldsymbol{\tau}) p_{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{\tau}) \log \sum_o p(o) p(\boldsymbol{s}, \boldsymbol{\tau}|o) \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s}$$
$$\approx \frac{1}{N} \sum_{(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \in \mathcal{D}} \tilde{W}(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \log \sum_{o \in \mathcal{O}} p(o) p(\boldsymbol{s}_i, \boldsymbol{\tau}_i|o). \tag{18}$$

To efficiently solve the maximization problem, we assume that an option policy is given by a Gaussian policy $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o) \sim \mathcal{N}(\boldsymbol{f}(\boldsymbol{s}), \boldsymbol{\Sigma})$, which is frequently assumed in RL (Deisenroth, Neumann, and Peters 2013; Pirotta, Restelli, and Bascetta 2013; Furmston and Barber 2012). The mean $\boldsymbol{f}(\boldsymbol{s})$ can be linear to the feature vector as $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{s})$ or represented by the output of a neural network. If we assume that the region of the context $d(\boldsymbol{s}|o)$ for which the option policy is responsible is Gaussian, $p(\boldsymbol{s}, \boldsymbol{\tau}|o) = d(\boldsymbol{s}|o)\pi(\boldsymbol{\tau}|\boldsymbol{s}, o)$ should also be Gaussian. Under these assumptions, the latent variable estimation is considered fitting a Gaussian mixture model. Even if a cluster of samples $\{\boldsymbol{s}_i, \boldsymbol{\tau}_i\}$ is not Gaussian in practice, it will be represented by a mixture of Gaussian option policies.

The log marginal likelihood maximization for the Gaussian mixture model fitting can be performed by the expectation-maximization (EM) algorithm in general. We can employ the variational Bayes expectation-maximization (VBEM) algorithm as well as the maximum likelihood (ML) EM algorithm (Bishop 2006). The advantage of the VBEM algorithm over the ML EM algorithm is that the use of the symmetric Dirichlet distribution as a prior of the mixing coefficient leads to a sparse solution. This property of VBEM is preferable in our hierarchical policy search since it is likely that clusters found by VBEM focus on separate modes of the density and that unnecessary clusters are automatically eliminated. Namely, we can obtain option policies corresponding to modes of the return function by using VBEM for the latent variable estimation.

To obtain various option policies, Daniel et al. (2016) employed a constraint to avoid the overlap between option policies and the number of the option policies was gradually reduced in the learning process. Likewise, Florensa, Duan, and Abbeel (2017) assumed that the number of the option policies is given by the user and modified the reward function based on mutual information in order to encourage visiting

new states. However, these constraints cannot prevent an option policy from averaging over multiple modes of the return function. On the contrary, we do not employ such explicit constraints on option policies in HPSDE. By using a variational approach for estimating the latent variable, we can obtain a sparse solution which covers all the given samples with the minimum number of option policies. Therefore, our approach can automatically determine the number of the option policies and each option policy obtained by HPSDE focuses on a separate single mode of the return function.

In practice, the dimensionality of $\boldsymbol{s}_i$ and $\boldsymbol{\tau}_i$ can be high and the shape of the cluster may not be Gaussian in the original feature space. In such a case, nonlinear feature mapping with dimensionality reduction, e.g., the Laplacian eigenmaps (Belkin and Niyogi 2003), can be used to perform the latent variable estimation properly. In addition, trajectory representations such as Dynamic Movement Primitives (DMPs) (Ijspeert, Nakanishi, and Schaal 2002) can also be used to represent a trajectory with a small number of parameters.

After estimating $p(o|\boldsymbol{s}_i, \boldsymbol{\tau}_i)$ for each sample, each option policy $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o)$ can be updated using an off-the-shelf policy search algorithm for learning a monolithic policy. In the next section, we describe the gating policy that selects the optimal option policy for a given context.

Selection of the Option Policy

When a context $\boldsymbol{s}$ is given by an environment and the option policies $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o)$ are learned, the role of the gating policy $\pi(o|\boldsymbol{s})$ is to identify the option policy that maximizes the expected return for a given context. Therefore, the gating policy is given by

$$\pi(o|\boldsymbol{s}) : o^* = \arg\max_{o \in \mathcal{O}} \mathbb{E}\left[R|\pi(\boldsymbol{\tau}|\boldsymbol{s}, o), \boldsymbol{s}, o\right], \tag{19}$$

where the conditional expectation $\mathbb{E}\left[R|\pi(\boldsymbol{\tau}|\boldsymbol{s}, o), \boldsymbol{s}, o\right]$ is given by

$$\mathbb{E}\left[R|\pi(\boldsymbol{\tau}|\boldsymbol{s}, o), \boldsymbol{s}, o\right] = \int \pi(\boldsymbol{\tau}|\boldsymbol{s}, o) R(\boldsymbol{s}, \boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau}. \tag{20}$$

Under the assumption that option policies are Gaussian, we explicitly estimate the expected return for each option policy $\mathbb{E}\left[R|\pi(\boldsymbol{\tau}|\boldsymbol{s}, o), \boldsymbol{s}, o\right]$. For this purpose, we approximate the return function with a Gaussian Process (GP) (Rasmussen and Williams 2005)

$$R(\boldsymbol{\tau}, \boldsymbol{s}) \sim \mathcal{GP}\left(m\left(\boldsymbol{z}\right), k\left(\boldsymbol{z}, \boldsymbol{z}'\right)\right), \tag{21}$$

where $\boldsymbol{z} = [\boldsymbol{\tau}^\top, \boldsymbol{s}^\top]^\top$ and $m(\boldsymbol{z})$ is the mean. For the kernel function $k\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right)$, we employ the squared exponential kernel:

$$k\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right) = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{z}_i - \boldsymbol{z}_j\|^2}{2l^2}\right) + \sigma_n^2 \delta_{\boldsymbol{z}_i \boldsymbol{z}_j}, \tag{22}$$

where l is the bandwidth of the kernel, σ_f^2 is the function variance and σ_n^2 is the noise variance. The hyperparameters $[l, \sigma_f, \sigma_n]$ can be tuned by maximizing the marginal likelihood using a gradient-based method (Rasmussen and Williams 2005).

Algorithm 1 Hierarchical Policy Search via Return-Weighted Density Estimation (HPSDE)

Input: the maximum number of the clusters $O_{\max}$
Initialize the option policies, e.g. random policy
Collect the rollout data $\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{\tau}_i, R_i)\}$ by performing the initial policy
repeat
  Compute the importance of each sample $\tilde{W}(\boldsymbol{s}_i, \boldsymbol{\tau}_i)$
  Estimate $p(o|\boldsymbol{s}_i, \boldsymbol{\tau}_i)$ through density estimation using the importance weight $\tilde{W}$
  Assign the samples $\{(\boldsymbol{s}_i, \boldsymbol{\tau}_i)\}$ to option $o_i^* = \arg\max p(o|\boldsymbol{s}_i, \boldsymbol{\tau}_i)$
  for each o **do**
    Train the oth policy using a policy search method
  end for
  Train the GP model to approximate the return function
  Select the option $o^* = \arg\max \mathbb{E}\left[R|\pi(\boldsymbol{\tau}|\boldsymbol{s}, o), \boldsymbol{s}, o\right]$
  Execute the rollout by following $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o^*)$
  Record the data $\mathcal{D}_{\text{new}} = \{(\boldsymbol{s}_j, \boldsymbol{\tau}_j, R_j)\}$
  Store the data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{\text{new}}$
until the task learned

If we assume that the context is given and a trajectory is drawn from a Gaussian option policy $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o) \sim \mathcal{N}(\boldsymbol{\mu}^o(\boldsymbol{s}), \boldsymbol{\Sigma}^o(\boldsymbol{s}))$, $\boldsymbol{z}$ follows the Gaussian distribution:

$$\boldsymbol{z} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{z}}, \boldsymbol{\Sigma}_{\boldsymbol{z}}\right), \tag{23}$$

where

$$\boldsymbol{\mu}_{\boldsymbol{z}} = \begin{bmatrix} \boldsymbol{\mu}^o(\boldsymbol{s}) \\ \boldsymbol{s} \end{bmatrix}, \boldsymbol{\Sigma}_{\boldsymbol{z}} = \begin{bmatrix} \boldsymbol{\Sigma}^o(\boldsymbol{s}) & 0 \\ 0 & 0 \end{bmatrix}. \tag{24}$$

To compute the conditional expectation of the return given the option and the context in (20), we need to compute the following marginal distribution:

$$p(R|\boldsymbol{\mu}_{\boldsymbol{z}}, \boldsymbol{\Sigma}_{\boldsymbol{z}}) = \int p(R|\boldsymbol{z}, \mathcal{D}) p(\boldsymbol{z}) d\boldsymbol{z}. \tag{25}$$

The above marginal can be approximated with a Gaussian, and its mean $\mathbb{E}[R|\pi(\boldsymbol{\tau}|\boldsymbol{s}, o), \boldsymbol{s}, o]$ and variance σ_R can be analytically computed (Girard et al. 2002; Candela and Girard 2002).

While the marginal distribution $p(o|\boldsymbol{s})$ obtained in the density estimation can be used as the gating policy in our framework, the gating policy presented in this section works better in practice as shown in the experimental result section. In the experiment, we used the UCB policy to encourage the exploration in the selection of option policies (Auer 2003).

Algorithm

We summarize the procedure of HPSDE in Algorithm 1. An open parameter of HSPDE is the maximum number of option policies $O_{\max}$. If sufficiently large $O_{\max}$ is given, the number of the option policies is automatically determined. If necessary, one can perform dimensionality reduction before the return-weighted density estimation step.

Connection to Policy Parameter Estimation

In prior work by Daniel et al. (2016), the parameter of the option policy $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o)$ needs to be estimated in order to estimate $p(o|\boldsymbol{s}, \boldsymbol{\tau})$. On the contrary, our latent variable estimation does not need to estimate $\pi(\boldsymbol{\tau}|\boldsymbol{s}, o)$, since our approach estimates the latent variable o by directly approximating the density $p^*(\boldsymbol{s}, \boldsymbol{\tau})$.

On the other hand, our approach is closely related to prior work on estimating policy parameters for learning a monolithic policy. Here, we consider a policy $\pi_{\boldsymbol{\theta}}$ parameterized by a vector $\boldsymbol{\theta}$. In episodic policy search, $D_{\mathrm{KL}}(p^*(\boldsymbol{s}, \boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\tau}))$ can be rewritten by using $p^*(\boldsymbol{s}, \boldsymbol{\tau}) = d(s)\pi^*(\boldsymbol{\tau}|\boldsymbol{s})$ and $p_\pi(\boldsymbol{s}, \boldsymbol{\tau}) = d(\boldsymbol{s})\pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s})$ as

$$
\begin{aligned}
&D_{\mathrm{KL}}(d(s)\pi^*(\boldsymbol{\tau}|\boldsymbol{s})||d(s)\pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s})) \\
&= \iint d(\boldsymbol{s})\pi^*(\boldsymbol{\tau}|\boldsymbol{s}) \log \frac{d(s)\pi^*(\boldsymbol{\tau}|\boldsymbol{s})}{d(s)\pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s} \\
&= \iint d(\boldsymbol{s}) W(\boldsymbol{s}, \boldsymbol{\tau}) \pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\boldsymbol{\tau}|\boldsymbol{s}) \log \frac{W(\boldsymbol{s}, \boldsymbol{\tau})\pi_{\boldsymbol{\theta}_{\mathrm{old}}}}{\pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s})} \mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{s} \\
&\approx \frac{1}{N} \sum_{(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \in \mathcal{D}} (-W(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \log \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s}) \\
&\qquad + W(\boldsymbol{s}, \boldsymbol{\tau}) \log W(\boldsymbol{s}, \boldsymbol{\tau}) \pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\boldsymbol{\tau}|\boldsymbol{s})) \,. \qquad (26)
\end{aligned}
$$

Since the second term is independent of $\boldsymbol{\theta}$, the minimizer of $D_{\mathrm{KL}}(d(s)\pi^*(\boldsymbol{\tau}|\boldsymbol{s})||d(s)\pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s}))$ can be obtained by maximizing the weighted log likelihood as

$$
\pi^*_{\boldsymbol{\theta}} = \arg\max \frac{1}{N} \sum_{(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \in \mathcal{D}} W(\boldsymbol{s}_i, \boldsymbol{\tau}_i) \log \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\boldsymbol{s}).
$$

This is exactly the episodic version of reward-weighted regression (eRWR) (Peters and Schaal 2007; Kober and Peters 2011). Thus, the option estimation in our approach is closely related to eRWR for estimating the policy parameter, although our option estimation does not require estimating the option policy parameters.

Experimental Results

To visualize the performance, we first evaluate HPSDE in toy problems and the puddle world task where the return functions are multi-modal. Subsequently, we show the experiments with the motion planning task for a robotic manipulator, which is a practical application of hierarchical RL. In the experiment, we evaluated variants of HPSDE: we implemented HPSDE using REPS and reward weighted regression (RWR) (Peters and Schaal 2007) for updating the option policies. REPS is a model-free policy search algorithm and constrains the KL divergence between the old and the updated policies in the policy update, although RWR does not have such a constraint in the policy update. The constraint of the KL divergence on the policy update is frequently used to achieve stable exploration (Deisenroth, Neumann, and Peters 2013). We can see how the constraint on the option policy update influences on the performance of HPSDE by comparing HPSDE with REPS and HPSDE with RWR. To evaluate the gating policy using a GP described in the previous section, we implemented a variant of HPSDE where the gating policy is represented by a softmax function, which is equivalent to the gating policy used by Daniel et al. (2016). In addition, we compare the performance of HPSDE with HiREPS (Daniel et al. 2016), which is considered as one of the state-of-the-art model-free hierarchical policy search methods. In the experiment, we assumed that a trajectory $\boldsymbol{\tau}$ can be represented by a trajectory parameter $\boldsymbol{\xi}$, and the task is to learn a policy $\pi(\boldsymbol{\xi}|\boldsymbol{s})$ that generates trajectory parameter instead of a raw representation of a trajectory $\boldsymbol{\tau}$. we used a Gaussian policy for an option policy given by

$$
\pi\left(\boldsymbol{\xi}|\boldsymbol{s}, o\right) = \mathcal{N}(\boldsymbol{\xi}|\boldsymbol{w}_o^\top \boldsymbol{\phi}(\boldsymbol{s}), \boldsymbol{\Sigma}_o), \qquad (27)
$$

where $\boldsymbol{w}_o$ and $\boldsymbol{\Sigma}_o$ are option policy parameters to be learned, and the mean of the policy $\boldsymbol{w}_o^\top \boldsymbol{\phi}(\boldsymbol{s})$ is linear to the feature function $\boldsymbol{\phi}(\boldsymbol{s})$. For computing the importance weight in (11), we used $f(R) = \exp(R)$. We performed each task 20 times to evaluate the variance of the achieved return. To deal with high dimensional and non-Gaussian data, we employed the Laplacian eigenmaps (Belkin and Niyogi 2003) for nonlinear dimensionality reduction.

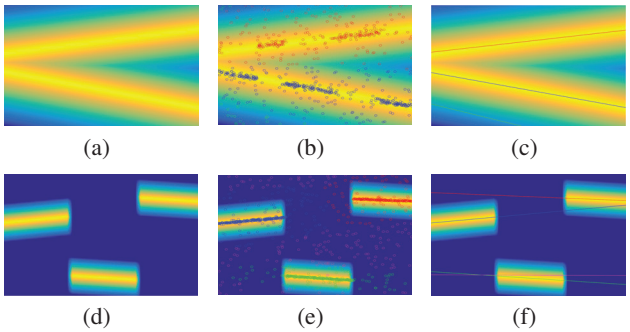


Figure 1: The toy problem using the return function with multiple modes. The horizontal axis shows the context, the vertical axis shows the trajectory parameter. The warmer color represents the higher return. (b) and (e) visualize samples collected by HPSDE. Learned option policies are visualized as lines as shown in (c) and (f). These results were obtained by HPSDE with REPS and the GP gating policy.

Toy Problem

We first evaluate the performance of HPSDE in toy problems. In this toy example, the context and the trajectory parameter are one-dimensional so that we can visualize the result intuitively. We performed evaluation using return functions with two and three modes, which are shown in Figure 1. For this task, we used a linear feature function $\boldsymbol{\phi}(\boldsymbol{s}) = [\boldsymbol{s}^\top, 1]^\top$ and set $O_{\max} = 10$ for HPSDE.

Figure 1 visualizes the samples collected in the learning process and policies learned by HPSDE. As shown, HPSDE identified the modes of the return function in both toy problems. Although our approach extracted option policies from sample clusters with low returns, it is rarely selected since the expected return is always lower than other policies.

The resulting return and the number of the learned policies are shown in Figure 2. HPSDE achieved higher returns

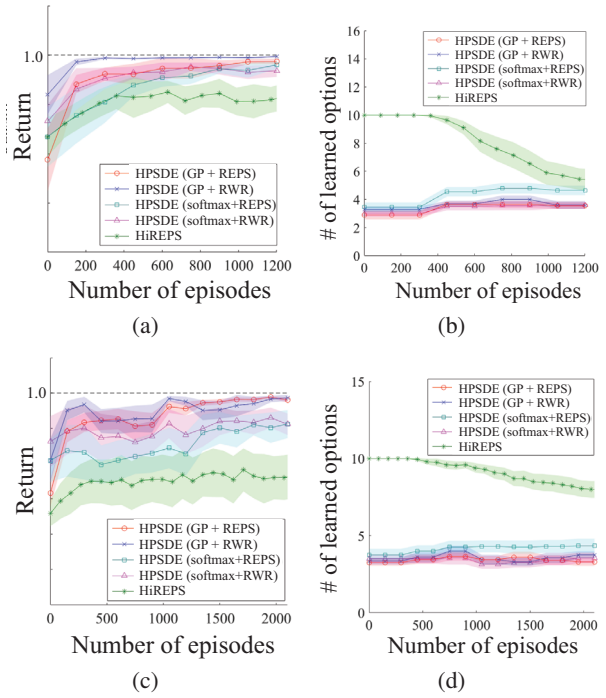


Figure 2: Results of the toy problems. (a) and (b) show the results with the return function that has two modes shown in Figure 1 (a)-(c), and (c) and (d) show the results with the return function that has three modes shown in Figure 1 (d)-(f).

than HiREPS in these toy tasks. In addition, although both HiREPS and HPSDE converge to comparable numbers of option policies, HPSDE optimizes the number of the option policies much faster. Since the policy updates of option policies in HiREPS and HPSDE with REPS are equivalent, this result indicates that the identification of the option policy structure in HPSDE is more efficient than that in HiREPS. In addition, although HiREPS cannot increase the number of the option policies in the learning process, our approach finds the optimal number of the option policies without such a limitation.

With regard to the gating policy, the gating policy with the return approximation using a GP outperforms the gating policy represented by a softmax function. With respect to the strategy for updating the option policies, the learning rate of HPSDE with RWR was comparable to that of HPSDE with REPS in the toy problems, although REPS was reported to achieve faster learning than RWR in many cases (Peters, Mülling, and Altun 2010). When a given sample distribution has multiple modes of the return function, the constraint on the KL divergence between the old and the updated policies prevents from jumping from the current mode to another mode. On the other hand, when a sample distribution given to a policy has just one mode of the return function, the constraint in the policy update does not clearly improve the learning performance. Since our framework successfully identifies the modes of the return function in these toy examples, the constraint on the KL divergence in the policy update does not make a clear difference of performance between REPS and RWR in HPSDE.

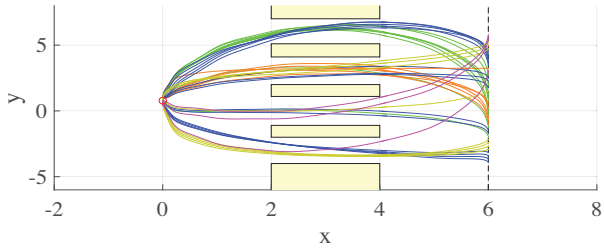


Figure 3: The puddle world task. Routes from the starting position to the goal positions are found by HPSDE. The starting position is fixed as shown as the red point and the goal positions are distributed on the line $x = 6$. Trajectories in the same color are generated by the same option policy.

Puddle World Task

We tested HPSDE with a variant of the puddle world task reported by Daniel et al. (2016) where a collision free trajectory needs to be planned in a continuous space. We used DMPs to represent trajectories in this task (Ijspeert, Nakanishi, and Schaal 2002). The arrangement of the puddles is shown in Figure 3, which is more complex than the task reported by Daniel et al. (2016). We assume that the starting position is fixed, and the goal position, which is given as a context of this task, is distributed on the line $x = 6$. A trajectory is represented with two DMPs, which represent x and y dimensions, respectively. In this task, the DMP for x dimension is fixed and the DMP for y dimension is optimized with HPSDE. We use 10 basis functions to represent a trajectory, and therefore HPSDE optimized 10 parameters of the DMP. The return function is designed such that passing through the puddle results in a negative return and a longer trajectory receives a less return, which encourages a shorter and collision-free trajectory. For this experiment, we used the squared exponential feature:

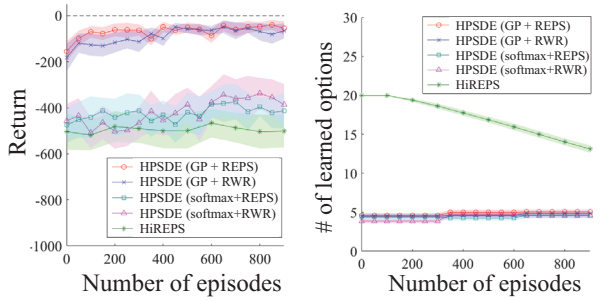
$$\phi_i(\boldsymbol{s}) = \exp\left((\boldsymbol{s} - \boldsymbol{s}_i)^\top \Lambda (\boldsymbol{s} - \boldsymbol{s}_i)\right), \qquad (28)$$

where Λ is a diagonal matrix that defines the bandwidth. This exponential feature enables us to represent a nonlinear policy (Daniel et al. 2016). We set $O_{\max} = 20$ for HPSDE.

As shown in Figure 3, HPSDE finds multiple solutions in the puddle world task. Figure 4 shows the resulting return and the number of the learned option policies. In this experiment, HiREPS falls into local optima and converges to poor performance. On the contrary, HPSDE achieves much higher performance by finding option policies corresponding to modes of the return functions. These results show that the learning of the option policy structure with HPSDE is more efficient than HiREPS.

Motion Planning for a Redundant Manipulator

Planning a motion of a redundant manipulator has been an open problem in robotics since there exists multiple solu-



(a) (b)

Figure 4: Results of the puddle world task. (a) returns and (b) the number of the learned option policies.

tions in the continuous space. We evaluated HPSDE with a motion planning problem for such a redundant manipulator in a simulation environment, developed based on V-REP (Rohmer, Singh, and Freese 2013). Figure 5 shows the simulation environment. A KUKA Light Weight Robot with 7 degrees of freedom is modeled in this simulation, and the task is to touch a desired point behind a pole. The goal point is distributed behind the pole, and the context of this task is given by the Cartesian coordinates of the goal position as $\boldsymbol{s} = [x, y, z]$. In this task, the objective of hierarchical RL is to optimize the final configuration of the robotic manipulator $\boldsymbol{q} \in \mathbb{R}^7$. The return is given by $R(\boldsymbol{q}) = -d(\boldsymbol{q}) - C(\boldsymbol{q})$, where $d(\boldsymbol{q})$ is the distance between the position of the end-effector and the desired position, and $C(\boldsymbol{q})$ is the cost of colliding with the pole for a given configuration $\boldsymbol{q}$. The collision cost is computed based on the cost function proposed by Zucker et al. (2013). We used the squared exponential feature as in the previous experiment. We set $O_{\max} = 10$ for HPSDE.

As shown in Figure 5, HPSDE found multiple policies to achieve the reaching task. The learning performance is shown in Figure 6. HPSDE with REPS and the GP gating policy demonstrates the best performance in this task. Although HiREPS also achieves performance comparable to some variants of HPSDE, it is necessary to heuristically specify the minimum and maximum numbers of the option policies and the parameter to delete the option policies in order to obtain the best performance of HiREPS for this task. On the contrary, an open parameter in HPSDE is just the maximum number of the option policies.

When the start and goal configurations are given, trajectory optimization methods developed in robotics such as CHOMP (Zucker et al. 2013) and TrajOpt (Schulman et al. 2014) can be used for motion planning. However, planning the desired configuration itself is often challenging since there exist multiple solutions in multi-dimensional and continuous space. The result in this work indicates that HPSDE can address such motion planning problems in robotics well.

Conclusion

We proposed the hierarchical policy search via return-weighted density estimation (HPSDE). To address the issue of determining the structure of the option policies, our approach reduces the problem of estimating the modes of the return function to the problem of estimating the return-weighted sample density with a mixture model. HPSDE automatically identifies the option policy structure, where each option policy corresponds to a single mode of the return function. The connection between the expected return maximization and the return-weighted density estimation is analytically shown in this study. The experimental results show that HPSDE outperforms a state-of-the-art method for hierarchical reinforcement learning, and that HPSDE can be used to solve motion planning problem for a redundant robotic manipulator. Although we employed the gating policy using GPs, it is hard to scale it to high dimensional data. In future work, we will extend the proposed approach so that high dimensional data such as image inputs can be incorporated. Additionally, we will extend the episodic HPSDE proposed in this paper to the learning of an action-state level policy and perform experiments with a real robot.

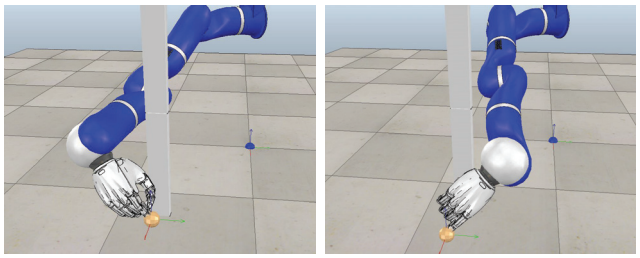
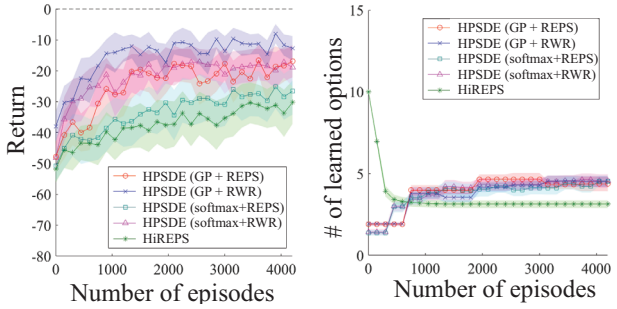


Figure 5: The reaching task with a robot with 7 degrees of freedom. Multiple postures to reach the desired point were learned by HPSDE. The orange sphere shows the goal point.



(a) (b)

Figure 6: Results of the motion planning task for a redundant manipulator. (a) returns and (b) the number of the learned option policies.

Acknowledgments

OT and MS were supported by KAKENHI 17H00757.

References

Auer, P. 2003. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* 3:397–422.

Bacon, P. L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Belkin, M., and Niyogi, P. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation* 15(6):1373–1396.

Bishop, C. M. 2006. *Pattern recognition and machine learning*. Springer.

Candela, J. Q., and Girard, A. 2002. Prediction at an uncertain input for gaussian processes and relevance vector machines – application to multiple-step ahead time-series forecasting. Technical report, Danish Technical University.

Cutkosky, M. R., and Howe, R. D. 1990. Human grasp choice and robotic grasp analysis. In Venkataraman, S. T., and Iberall, T., eds., *Dextrous Robot Hands*. Springer-Verlag New York, Inc. 5–31.

Daniel, C.; Neumann, G.; Kroemer, O.; and Peters, J. 2016. Hierarchical relative entropy policy search. *Journal of Machine Learning Research* 17:1–50.

Dayan, P., and Hinton, G. 1997. Using expectation-maximization for reinforcement learning. *Neural Computation* 9:271–278.

Deisenroth, M. P.; Neumann, G.; and Peters, J. 2013. A survey on policy search for robotics. *Foundations and Trends in Robotics* 388–403.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Florensa, C.; Duan, Y.; and Abbeel, P. 2017. Stochastic neural networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Furmston, T., and Barber, D. 2012. A unifying perspective of parametric policy search methods for markov decision processes. In *Advances in Neural Information Processing Systems (NIPS)*.

Girard, A.; Rasmussen, C. E.; Candela, J. Q.; and Murray-Smith, R. 2002. Gaussian process priors with uncertain inputs – application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems (NIPS)*.

Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ijspeert, A. J.; Nakanishi, J.; and Schaal, S. 2002. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*.

Kober, J., and Peters, J. 2011. Policy search for motor primitives in robotics. *Machine Learning* 84:171–203.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016a. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17(39):1–40.

Levine, S.; Pastor, P.; Krizhevsky, A.; and Quillen, D. 2016b. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. In *Proceedings of International Symposium on Experimental Robotics (ISER)*.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Napier, J. R. 1956. The prehensile movements of the human hand. *Journal of Bone and Joint Surgery* 38-B(4):902–13.

Osa, T.; Peters, J.; and Neumann, G. 2016. Experiments with hierarchical reinforcement learning of multiple grasping policies. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*.

Peters, J., and Schaal, S. 2007. Reinforcement learning by reward-weighted regression for operational space control. In *International Conference on Machine Learning (ICML)*.

Peters, J.; Mülling, K.; and Altun, Y. 2010. Relative entropy policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Pirotta, M.; Restelli, M.; and Bascetta, L. 2013. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems (NIPS)*.

Rasmussen, C. E., and Williams, C. K. I. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Rohmer, E.; Singh, S. P. N.; and Freese, M. 2013. V-rep: a versatile and scalable robot simulation framework. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Schulman, J.; Duan, Y.; Ho, J.; Lee, A.; Awwal, I.; Bradlow, H.; Pan, J.; Patil, S.; Goldberg, K.; and Abbeel, P. 2014. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9):1251–1270.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, P. 2015. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Ziebart, B. 2010. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. Ph.D. Dissertation, Carnegie Mellon University.

Zucker, M.; Ratliff, N.; Dragan, A.; Pivtoraiko, M.; Klingensmith, M.; Dellin, C.; Bagnell, J. A.; and Srinivasa, S. 2013. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32:1164–1193.