Layered Direct Policy Search for Learning Hierarchical Skills

Felix End[1], Riad Akrour[2], Jan Peters[3] and Gerhard Neumann[4]

***Abstract*— Solutions to real world robotic tasks often require complex behaviors in high dimensional continuous state and action spaces. Reinforcement Learning (RL) is aimed at learning such behaviors but often fails for lack of scalability. To address this issue, Hierarchical RL (HRL) algorithms leverage hierarchical policies to exploit the structure of a task. However, many HRL algorithms rely on task specific knowledge such as a set of predefined sub-policies or sub-goals. In this paper we propose a new HRL algorithm based on information theoretic principles to autonomously uncover a diverse set of sub-policies and their activation policies. Moreover, the learning process mirrors the policys structure and is thus also hierarchical, consisting of a set of independent optimization problems. The hierarchical structure of the learning process allows us to control the learning rate of the sub-policies and the gating individually and add specific information theoretic constraints to each layer to ensure the diversification of the sub-policies. We evaluate our algorithm on two high dimensional continuous tasks and experimentally demonstrate its ability to autonomously discover a rich set of sub-policies.**

I. Introduction

Reinforcement Learning (RL) algorithms are aimed at solving complex robotic tasks autonomously by interacting with the environment; circumventing the reliance on hand coded policies and human intervention. However, for many tasks 'flat' RL algorithms are ill suited as they are unable to uncover and exploit the structure of high dimensional state and action spaces. For example, to play table tennis, a robot needs to be able to execute several strokes such as backhand and forehand strokes. From a flat RL perspective this task requires to find an appropriate trade-off between i) policy complexity to encompass the various strokes and ii) data efficiency as simpler policies are more data efficient. However, a Hierarchical RL (HRL) approach can use hierarchy to decompose such complex behavior into simple sub-policies for each stroke and a simple gating network to choose the correct stroke. Such a structure also allows learning multiple kinds of forehand and backhands strokes. Having multiple solutions for the same sub-task gives us backup solution should the optimal sub-policy become inadequate (e.g. because of wear or damage to a robot). However, to use the solutions as effective backup we need them to be diverse.

[1]Felix End is a computer science student at the TU Darmstadt, Germany felix.end@stud.tu-darmstadt.de

[2]Riad Akrour is with the CLAS lab, TU Darmstadt, Germany akrour@ias.tu-darmstadt.de

[3]Jan Peters is with the IAS lab, TU Darmstadt, Germany and the Max Planck Institute for Intelligent Systems, Tuebingen, Germany peters@ias.tu-darmstadt.de

[4]Gerhard Neumann is with the School of Computer Science, University of Lincoln, United Kingdom and the CLAS lab, TU Darmstadt, Germany gneumann@lincoln.ac.uk

Many HRL methods have been proposed in order to reduce the complexity of the task [1]–[4]. The HAM framework [5] can learn complex hierarchical sub-routines. The MAXQ framework uses hierarchically sub-tasks with sub-goals to learn a hierarchical policy. Whereas HAM and MAXQ are for discrete domains the work of Morimoto et al. [3] and the Hierarchical Policy Gradient Algorithms [6] have shown ways to decompose a task into sub-tasks and learning a hierarchical policy for these sub-tasks in continuous domains. A disadvantage of these algorithms is that their decomposition in sub-tasks or their policy needs to be tailored to the task consequently they cannot be easily applied to new tasks.

Other hierarchical approaches such as Hierarchical Relative Entropy Policy Search (HiREPS) [7] use more flexible hierarchical policies that are not handcrafted for one specific task. The policy HiREPS optimizes is a mixture model consisting of sub-policies which solve the task and a gating network which chooses which sub-policy to use. A disadvantage of HiREPS is that it is not hierarchical in the learning process. Impeding the control of properties like diversity and the individual learning rates of the gating and sub-policies. This lack of control often leads HiREPS to stop optimizing all sub-policies except one.

In this paper, we propose a new episodic hierarchical policy search algorithm named Layered Direct Policy Search (LaDiPS). LaDiPS combines the advantages of a hierarchical policy and a hierarchical learning process. The policy that LaDiPS uses, is a Gaussian mixture model. The mixture model is optimized in two layers which mimics the hierarchy of the policy. The first layer optimizes the mixture components also called sub-policies while the second layer optimizes the gating. This decomposition allows us to control the exploration for each sub-policy and the gating individually and also gives us a way to control the diversity of the mixture model. To optimize the sub-policies and gating, we use information theoretic policy updates.

II. Preliminaries

This section introduces the formal notations (Sec. II-A), compares LaDiPS to other HRL algorithms (Sec. II-B) and describes the HiREPS algorithm (Sec. II-C) in more details. The HiREPS algorithm bears similarities with LaDiPS and its presentation better highlights the contributions of this paper.

A. Problem Formulation and Notation

LaDiPS is a Policy Search (PS) algorithm for contextual tasks and follows the formalism described in [8]. A context vector $\boldsymbol{x}$ characterizes the initial configuration of the task and is drawn at the start of each episode from a fixed probability

distribution. For a policy parameter $\boldsymbol{\theta} \in \Theta$ (e.g. $\boldsymbol{\theta}$ can be the gains of a PD-controller or the paramters of a motion primitive such as [9]), $r(\boldsymbol{x}, \boldsymbol{\theta})$ gives the reward of choosing parameter $\boldsymbol{\theta}$ under context $\boldsymbol{x}$. The goal of PS is to find a stochastic policy $\pi(\boldsymbol{\theta}|\boldsymbol{x})$ that maximizes the policy return $J(\pi) = \mathbb{E}_{\boldsymbol{\theta}\sim\pi(\boldsymbol{\theta}|\boldsymbol{x})}[r(\boldsymbol{x}, \boldsymbol{\theta})]$.

LaDiPS is an iterative algorithm. At iteration t, N pairs $(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]})$ are generated by observing the context $\boldsymbol{x}^{[i]}$ and drawing parameter $\boldsymbol{\theta}^{[i]}$ from policy $\pi_t(\cdot|\boldsymbol{x}^{[i]})$. The samples $D = \{(\boldsymbol{\theta}^{[i]}, \boldsymbol{x}^{[i]}, r(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]}))|i = 1, \ldots, N\}$ are used to update the old policy π_t to the new policy π_{t+1}. In the remaining paper we will denote π_t and probabilities derived from π_t by q, π_{t+1} by π and probabilities derive from π_{t+1} by p.

B. Related Work

Out of the HRL algorithms MAXQ [5], Hierarchies of Machines (HAM) [10], Morimoto et al.'s algorithm [3] and Hierarchical Policy Gradient Algorithms (HPG) [6], HPG is the most closely related to LaDiPS as HPG is aimed at solving continuous problems (unlike MAXQ and HAM) and it is more general than the work of Morimoto et al. HPG is a framework for solving complex tasks with hierarchical policies. The task has to be divided into sub-tasks which have their own sub-goals. The lower level sub-tasks are learned with policy gradient methods whereas the high level sub-tasks are learned with value function based methods. Defining sub-tasks by hand allows for the use of more complex hierarchies than the hierarchy of LaDiPS. However, a new hierarchy has to be defined for every new task and defining useful sub-goals requires that the programmer already knows the structure of a good solution. Additionally, good solutions that the programmer did not think of cannot be found if these solutions do not optimize the sub-goals. In the LaDiPS algorithm we do not define sub-tasks which means the algorithm can find the best sub-policies for the overall tasks.

The mixture of motor primitives (MoMP) [4] is a HRL algorithm that uses a very different hierarchical approach. MoMP first learns sub-policies (called motor primitives) and then learns a gating network to generate new movements by combining the sub-policies. There are two important ways in which MoMP is different from LaDiPS. First, MoMP mixes the sub-policies whereas LaDiPS only uses the gating network to choose one of the sub-policies. The second difference is that MoMP learns the lower layer of the hierarchy (the sub-policies) independently of the gating. After the sub-policies are learned, the upper layer (the gating network) is learned independently. A disadvantage of learning the upper layer with a fixed lower layer is that the lower layer cannot adapt to better suit the needs of the higher layer of the hierarchy. In contrast, LaDiPS learns both layers simultaneously which allows the sub-policies and gating network to adapt to each other.

C. Hierarchical Relative Entropy Policy Search

The Hierarchical Relative Entropy Policy Search (HiREPS) [7] algorithm is closely related to LaDiPS. Both algorithms learn a mixture model policy and use information theoretic principles to update this policy. However, the crucial difference between HiREPS and LaDiPS is how the policy is updated. While, HiREPS optimizes a single constrained optimization problem, grouping the gating network and the sub-policies together, LaDiPS adopts a more decentralized approach by optimizing each layer separately. The benefits of disentangling the optimization problem and breaking it into smaller sub-problems are two fold. First, constraints on the information loss can be added to each layer individually allowing different learning rates for the gating network and the sub-policies. Secondly, additional information theoretic constraints can be integrated to the layers in order to better control the exploration of the sub-policies and to force a number of sub-policies to stay 'active' (i.e., have nonzero probability), allowing the algorithm to explore additional modes of the reward function. HiREPS lacks such control mechanisms causing it to frequently only learn a single sub-policy as the activation of the other sub-policies decreased to zero.

To update its policy HiREPS optimizes an optimization problem. The objective of the optimization problem is maximizing the average reward of the policy. This objective is optimized with four constraints of which the first two are interesting to us. The first constraint bounds the information loss between the current policy p and the previous policy q; a common practice in the RL community [11]–[13] and is also used by LaDiPS. The second constraint ensures sub-policies are diverse. However, this constraint is often not effective because HiREPS lacks a way to ensure that multiple sub-policies retain some probability. When all sub-policies except one converge to zero probability during the first few iterations of training, then only one sub-policy ends up being properly trained, making the diversity of the other poorly trained sub-policies irrelevant.

In LaDiPS diversity is enforced differently. LaDiPS learns a reward model for each sub-policy with weighted ridge regression. The weights are chosen to ensure that the reward models are diverse. Additionally, LaDiPS uses the decentralized structure of its layers to add a constraint to the gating layer which controls how many sub-policies have to retain some probability. The combination of this constraint with the diverse reward models allows LaDiPS to learn diverse solutions more reliably than HiREPS.

III. Layered Direct Policy Search (LaDiPS)

This section explains LaDiPS by first giving an overview of the algorithm (Sec. III-A) before elaborating on the optimization process of the sub-policies (Sec. III-B) and the gating network (Sec. III-C).

A. LaDiPS Overview

LaDiPS is a reinforcement learning algorithm to optimize a mixture model policy, given by

$$\pi(\boldsymbol{\theta}|\boldsymbol{x}) = \sum_{j=1}^{K} \pi_j(\boldsymbol{\theta}|\boldsymbol{x})\pi(j|\boldsymbol{x}).$$

The gating $\pi(j|\boldsymbol{x})$ selects the sub-policy index j. As gating network we use a softmax policy, given by

$$\pi(j|\boldsymbol{x}) = \frac{\exp\left(\boldsymbol{\rho}_j^T \boldsymbol{\phi}(\boldsymbol{x})\right)}{\sum_{l=1}^{K} \exp\left(\boldsymbol{\rho}_l^T \boldsymbol{\phi}(\boldsymbol{x})\right)}.$$

The sub-policy $\pi_j(\boldsymbol{\theta}|\boldsymbol{x})$ selects the parameter vector $\boldsymbol{\theta}$. For the sub-policy we use

$$\pi_j(\boldsymbol{\theta}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_j(\boldsymbol{x}), \boldsymbol{\Sigma}_j),$$

where the mean of the Gaussian mixture component is given by

$$\boldsymbol{\mu}_j(\boldsymbol{x}) = \boldsymbol{\Upsilon}_j^T \boldsymbol{\psi}(\boldsymbol{x}).$$

The functions $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\psi}(\boldsymbol{x})$ are a feature functions. The weights of the softmax policy $\boldsymbol{\rho}_j$, the weights of the Gaussian $\boldsymbol{\Upsilon}_j$ and the covariance $\boldsymbol{\Sigma}_j$ are the parameters we want to learn. The number of sub-policies K is not learned.

LaDiPS optimizes each layer of the policy individually. The optimization consist of the same two steps for both layers. First, a reward model is learned for each policy. Second, the reward models are used to update the policies of the layer. The two layers influence each other through the sample weights that are used for the reward estimation. The probability $q(j|\boldsymbol{x})$ is used in the estimation of the sub-policy reward models and the probability $\pi_j(\boldsymbol{\theta}|\boldsymbol{x})$ is used in the estimation of the gating reward model. The whole optimization process with its different steps can be seen in Fig. 1. The following sections will explain the different steps in more detail.

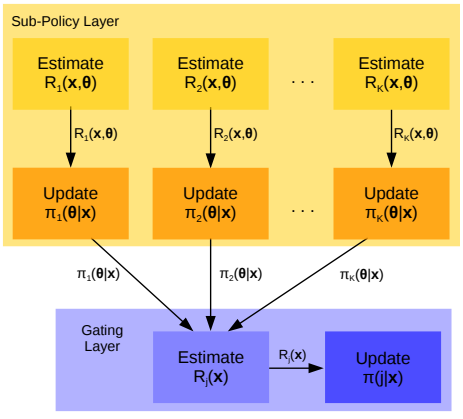

Fig. 1: LaDiPS overview showing the sub-policy layer (yellow, Sec. III-B) and gating layer (blue, Sec. III-C). Both layers consist of the same two steps which are reward model estimation (lighter color) and policy updates (darker color). The two layers influence each other through the probability from the previous iteration $q(j|\boldsymbol{x})$, which is used in the sub-policy reward $R_j(\boldsymbol{x}, \boldsymbol{\theta})$ estimation (not displayed) and through the updated sub-policies $\pi_j(\boldsymbol{\theta}|\boldsymbol{x})$, which are used in the gating reward $R_j(\boldsymbol{x})$ estimation.

B. Learning on the Sub-Policy Layer

The sub-policy optimization is an extension of Model-Based Relative Entropy Stochastic Search (MORE) [14]. Both algorithms consist of the two steps: first estimating a reward model and second updating the (sub-)policy. The main differences between the sub-policy optimization and MORE are that the sub-policy optimization is formulated for contextual problems and that it optimizes multiple sub-policies by doing both reward model estimation and sub-policy update for each sub-policy individually. In comparison, MORE is not context dependent and only estimates one reward model and updates one policy. To explain the sub-policy optimization of LaDiPS, we introduce the policy update step first to highlight why we estimate a reward function and explain the reward estimation step second.

Policy Update. To update the policy, we solve the optimization problem given by

$$\max_{\pi_j(\boldsymbol{\theta}|\boldsymbol{x})} \iint \pi_j(\boldsymbol{\theta}|\boldsymbol{x}) p(\boldsymbol{x}|j) R_j(\boldsymbol{x}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\boldsymbol{x}, \tag{1}$$

$$\text{s.t.} \quad \xi \geq \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x}|j)} \left[\mathrm{KL}\left(\pi_j(\boldsymbol{\theta}|\boldsymbol{x}) || q(\boldsymbol{\theta}|\boldsymbol{x}, j)\right)\right], \tag{2}$$

$$\beta \leq \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x}|j)} \left[\mathrm{H}\left[\pi_j(\boldsymbol{\theta}|\boldsymbol{x})\right]\right], \tag{3}$$

$$1 = \int \pi_j(\boldsymbol{\theta}|\boldsymbol{x}) \, \mathrm{d}\boldsymbol{\theta} \quad \text{for all contexts } \boldsymbol{x}. \tag{4}$$

The term (1) defines the average reward maximized by sub-policy j. Constraint (2) limits the information loss w.r.t. to the previous sub-policy q_j. The information loss is calculated with the Kullback-Leibler divergence $\mathrm{KL}(\cdot||\cdot)$. Constraint (3) controls how much the sub-policy explores in each iteration. The parameter β is calculated from the previous policy q_j with

$$\beta = \mathrm{H}[q(\boldsymbol{\theta}|\boldsymbol{x}, j)] - h,$$

where h is a hyper parameter that is chosen such that the entropy $\mathrm{H}[\cdot]$ does not decrease too fast and consequently the policy only slowly reduces the amount of exploration. Constraint (4) ensures that the policy is correctly normalized.

The optimization problem is solved with the Lagrangian multipliers method [15]. The dual function is given by

$$g(\eta_j, \omega_j) = \xi\eta_j - \beta\omega_j + \int p(\boldsymbol{x}|j) \ln I_j(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$$

where

$$I_j(\boldsymbol{x}) = \int q(\boldsymbol{\theta}|\boldsymbol{x}, j)^{\frac{\eta_j}{\eta_j+\omega_j}} \exp\left(\frac{R_j(\boldsymbol{x}, \boldsymbol{\theta})}{\eta_j + \omega_j}\right) \mathrm{d}\boldsymbol{\theta}.$$

The parameter η_j is the Lagrangian multiplier to constraint (2) and the parameter ω_j is the Lagrangian multiplier to constraint (3). The Lagrangian multiplier of the last constraint (4) cancel out by using the constraint in the derivation.

Note that $I_j(\boldsymbol{x})$ is actually $I_j(\boldsymbol{x}, \eta_j, \omega_j)$, however we suppress the dependence on η_j and ω_j to shorten the equations. The same is true for the functions $A_j(\boldsymbol{x}), B_j, \boldsymbol{F}_j, \boldsymbol{f}_j(\boldsymbol{x})$ which are introduced in the following paragraph.

We can calculate the integral $I_j(\boldsymbol{x})$ analytically for our Gaussian sub-policies if we use a reward model $R_j(\boldsymbol{x}, \boldsymbol{\theta})$ that is quadratic in parameters and context features $(\boldsymbol{\theta}, \boldsymbol{\psi}(\boldsymbol{x}))$. To keep the parameters of the reward model low we use linear features $\boldsymbol{\psi}(\boldsymbol{x}) = (\boldsymbol{x}^T, 1)^T$. Using these features and the aforementioned reward function model the analytic solution of the integral is given by

$$I_j(\boldsymbol{x}) = \exp(A_j(\boldsymbol{x}) + B_j)$$

with

$$\begin{aligned} A_j(\boldsymbol{x}) &= \boldsymbol{f}_j(\boldsymbol{x})^T \boldsymbol{F}_j \boldsymbol{f}_j(\boldsymbol{x}) - \eta_j \boldsymbol{\mu}_j(\boldsymbol{x})^T \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\mu}_j(\boldsymbol{x}) \\ &\quad + 2c_j(\boldsymbol{x}), \\ B_j &= -\eta_j \ln\left(|2\pi \boldsymbol{\Sigma}_j|\right) \\ &\quad + (\eta_j + \omega_j) \ln\left(|2\pi(\eta_j + \omega_j)\boldsymbol{F}_j|\right), \end{aligned}$$

$$\begin{aligned} \boldsymbol{F}_j &= (\eta_j \boldsymbol{\Sigma}_j^{-1} - 2\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j)^{-1}, \\ \boldsymbol{f}_j(\boldsymbol{x}) &= \eta_j \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\Upsilon}_j^T \boldsymbol{\psi}(\boldsymbol{x}) + 2\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{x}}^j \boldsymbol{x} + \boldsymbol{r}_{\boldsymbol{\theta}}^j. \end{aligned}$$

The matrix $\boldsymbol{\Sigma}_j$ is the variance of the old distribution $q(\boldsymbol{\theta}|\boldsymbol{x}, j)$ and the vector $\boldsymbol{\mu}_j(\boldsymbol{x})$ is its mean. The term $c_j(\boldsymbol{x})$ is independent from the Lagrangian multipliers which means we can ignore it in the dual function. The terms $\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j$ and $\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{x}}^j$ come from the quadratic reward model. The reward model is given by

$$\begin{aligned} R_j(\boldsymbol{x}, \boldsymbol{\theta}) &= \boldsymbol{\theta}^T \boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j \boldsymbol{\theta} + 2\boldsymbol{\theta}^T \boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{x}}^j \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{x}}^j \boldsymbol{x} \\ &\quad + \boldsymbol{\theta}^T \boldsymbol{r}_{\boldsymbol{\theta}}^j + \boldsymbol{x}^T \boldsymbol{r}_{\boldsymbol{x}}^j + \varrho^j. \end{aligned} \tag{5}$$

If we approximate $p(\boldsymbol{x}|j)$ with a Gaussian distribution, we can also compute the integral over $\boldsymbol{x}$ in g analytically because $A_j(\boldsymbol{x}) + B_j$ is a quadratic function in $\boldsymbol{x}$.

The dual function g is minimized for $\eta_j > 0$ and $\omega_j > 0$. Since the dual function is convex [15], we can minimize it efficiently with a convex optimizer. Once η_j and ω_j are obtained the policy is updated by maximizing the Lagrangian of the optimization problem w.r.t. the policy. The new policy is again a Gaussian distribution with mean $\boldsymbol{F}_j \boldsymbol{f}_j(\boldsymbol{x})$ and variance $(\eta_j + \omega_j)\boldsymbol{F}_j$.

Reward Estimation. We want each quadratic reward model (5) to be accurate for those samples which have high probability to be generated by its corresponding sub-policy j. To make the reward model adapt to its sub-policy, we minimize the expected distance between the true reward and the model, given by

$$\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{\theta})\sim q(\boldsymbol{x},\boldsymbol{\theta}|j)} [(r(\boldsymbol{x}, \boldsymbol{\theta}) - R_j(\boldsymbol{x}, \boldsymbol{\theta}))^2].$$

However, we also want to use the samples of all sub-policies which we can achieve with importance weighting. The resulting objective function is given by

$$\begin{aligned} &\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{\theta})\sim q(\boldsymbol{x},\boldsymbol{\theta}|j)} [(r(\boldsymbol{x}, \boldsymbol{\theta}) - R_j(\boldsymbol{x}, \boldsymbol{\theta}))^2] \\ &\approx \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{q(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]}|j)}{q(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]})} \left[r(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]}) - R_j(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]})\right]^2, \end{aligned} \tag{6}$$

where D is the data introduced in section II-A. Looking at the weights $q(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]}|j)/q(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]})$, we see that samples that are specific to only one sub-policy will have a high enumerator and relatively low denominator. These weights cause the sub-policies to diversify as each reward model $R_j(\boldsymbol{x}, \boldsymbol{\theta})$ focuses more on rewards that are unique to its corresponding sub-policy. However, each sub-policy only diversifies w.r.t. the other sub-policies from the previous iteration. To make the sub-policies diversify w.r.t. each other we optimize the sub-policies in sequence, which allows us to compute the probability weighting for the next sub-policy based on the already updated sub-policies.

Additionally, we also multiply

$$b^i = \frac{1}{\left|r^{[i]} - \max_i(r^{[i]}) + 0.01(\max_i(r^{[i]}) - \min_i(r^{[i]}))\right|}$$

as extra weightings inside the sum (6) to make the reward model adapt more to samples with high reward as these samples are more important. The term $r^{[i]}$ is the reward $r(\boldsymbol{x}^{[i]}, \boldsymbol{\theta}^{[i]})$. We also use an extra penalty term for high parameters of $R_j(\boldsymbol{x}, \boldsymbol{\theta})$, which makes the optimization an instance of weighted ridge regression.

Since the reward model is used to update the covariance matrix $(\eta_j + \omega_j)\boldsymbol{F}_j$ of the new policy, the block matrix $\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j$ of the reward model needs to be negative semi-definite. To ensure that $\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j$ is negative semi-definite, we replace the positive eigenvalues of the quadratic part of $R_j(\boldsymbol{x}, \boldsymbol{\theta})$, given by

$$\boldsymbol{Q}^j = \begin{pmatrix} \boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j, \boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{x}}^j \\ \boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{x}}^j, \boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{x}}^j \end{pmatrix}$$

with zero. With the new quadratic part $\boldsymbol{Q}^j$ fixed, we maximize the expectation again for $\boldsymbol{r}_{\boldsymbol{\theta}}^j$, $\boldsymbol{r}_{\boldsymbol{x}}^j$ and ϱ^j. The new parameters give us a negative semi-definite reward model. Theoretically, it would be enough to only remove the positive eigenvalues from $\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j$, however, removing the positive eigenvalues of $\boldsymbol{Q}^j$ has shown better results in experiments.

C. Learning on the Gating Policy Layer

The gating optimization problem also consists of a policy update and a reward estimation step. Both steps are similar to their counterparts in the sub-policy optimization from section III-B. The main differences for the policy update are the softmax policy instead of the Gaussian policy and the use of the entropy constraint parameter. While for the sub-policy optimization the entropy constraint parameter β is decreased, the corresponding parameter of the gating optimization α stays constant over all iterations. The main differences for the reward estimation are the use of a different reward model which does not depend on $\boldsymbol{\theta}$ and that we do not need to enforce certain constraints on the reward model like the negative semi-definite constraint on $\boldsymbol{Q}_{\boldsymbol{\theta}\boldsymbol{\theta}}^j$.

Policy Update. The policy is updated by solving the optimization problem given by

$$\max_{\pi(j|\boldsymbol{x})} \int \sum_{j=1}^{K} \pi(j|\boldsymbol{x}) p(\boldsymbol{x}) R_j(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \tag{7}$$

$$\text{s.t.} \quad \varepsilon \geq \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x})} [\mathrm{KL}\left(\pi(j|\boldsymbol{x}) || q(j|\boldsymbol{x})\right)], \tag{8}$$

$$\alpha \leq \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x})} [\mathrm{H}[\pi(j|\boldsymbol{x})]], \tag{9}$$

$$1 = \sum_{j=1}^{K} \pi(j|\boldsymbol{x}) \quad \text{for all contexts } \boldsymbol{x}. \tag{10}$$

Equation (7), (8) and (10) fulfill the same role as (1), (2) and (4) in section III-B.

However, (9) is used very differently to (3). Whereas β is decreased over the iterations to slowly reduce exploration, α is used to control how many sub-policy indices have to retain some probability and consequently α is not changed over the iterations. The larger α is, the more sub-policy indices have to have some probability. The biggest value α can be set to is $\ln(K)$, where K is the total number of sub-policies used. At this point the constraint (3) can only be fulfilled if the gating network is the uniform distribution.

The optimization problem is solved with the method of Lagrangian multipliers [15], which yields the solution

$$\pi(j|\boldsymbol{x}) \propto q(j|\boldsymbol{x})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{R_j(\boldsymbol{x})}{\eta+\omega}\right).$$

The parameter η is the Lagrangian multiplier to the KL-divergence constraint (8) and the parameter ω is the Lagrangian multiplier to the entropy constraint (9). The Lagrangian multipliers of the last constraints (10) cancel out.

Since $q(j|\boldsymbol{x})$ has the form

$$q(j|\boldsymbol{x}) = \frac{\exp\left({\boldsymbol{\rho}_j}^T \boldsymbol{\phi}(\boldsymbol{x})\right)}{\sum_l^K \exp\left({\boldsymbol{\rho}_l}^T \boldsymbol{\phi}(\boldsymbol{x})\right)},$$

we can compute the new policy $\pi(j|\boldsymbol{x})$ directly if we choose $R_j(\boldsymbol{x}) = \bar{\boldsymbol{\rho}}_j^T \boldsymbol{\phi}(\boldsymbol{x})$, where $\boldsymbol{\phi}$ is the same feature function used for $q(j|\boldsymbol{x})$. For this reward model, the direct update for the new policy $\pi(j|\boldsymbol{x})$ is

$$\pi(j|\boldsymbol{x}) = \frac{\exp\left({\boldsymbol{\rho}_j^*}^T \boldsymbol{\phi}(\boldsymbol{x})\right)}{\sum_l^K \exp\left({\boldsymbol{\rho}_l^*}^T \boldsymbol{\phi}(\boldsymbol{x})\right)}$$

with

$$\boldsymbol{\rho}_j^* = \frac{\eta \boldsymbol{\rho}_j + \bar{\boldsymbol{\rho}}_j}{\eta+\omega}.$$

The variable $\boldsymbol{\rho}_j$ is the parameter from the old distribution $q(j|\boldsymbol{x})$. The variable $\bar{\boldsymbol{\rho}}_j$ is the feature weight of $R_j(\boldsymbol{x})$.

The parameters η and ω are obtained by minimizing the dual function

$$g(\eta,\omega) = \varepsilon\eta - \alpha\omega + (\eta+\omega) \int p(\boldsymbol{x}) \ln A(\boldsymbol{x},\eta,\omega) \, \mathrm{d}\boldsymbol{x}$$

with

$$A(\boldsymbol{x},\eta,\omega) = \sum_j q(j|\boldsymbol{x})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{R_j(\boldsymbol{x})}{\eta+\omega}\right)$$

for $\eta > 0$ and $\omega > 0$. The integral over $p(\boldsymbol{x}) \ln A(\boldsymbol{x},\eta,\omega)$ is approximated with samples. As before, the dual function g is convex, therefore we can minimize it efficiently.

Reward Estimation. We want the reward model given by

$$R_j(\boldsymbol{x}) = \bar{\boldsymbol{\rho}}_j^T \boldsymbol{\phi}(\boldsymbol{x})$$

to be accurate for samples which have a high probability to be generated by the gating network. To achieve this adaptation, we can use the same minimization which we used for the Reward Estimation in section III-B, given by

$$\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{\theta})\sim p(\boldsymbol{x},\boldsymbol{\theta}|j)} [(r(\boldsymbol{x},\boldsymbol{\theta}) - R_j(\boldsymbol{x}))^2]$$
$$\approx \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{p(\boldsymbol{x}^{[i]},\boldsymbol{\theta}^{[i]}|j)}{p(\boldsymbol{x}^{[i]},\boldsymbol{\theta}^{[i]})} \left[r(\boldsymbol{x}^{[i]},\boldsymbol{\theta}^{[i]}) - R_j(\boldsymbol{x}^{[i]})\right]^2.$$

However, in our experiments we have seen that using $\pi_j(\boldsymbol{\theta}^{[i]}|\boldsymbol{x}^{[i]},j)$ as weights for the gating reward yields better results than using $p(\boldsymbol{x}^{[i]},\boldsymbol{\theta}^{[i]}|j)/p(\boldsymbol{x}^{[i]},\boldsymbol{\theta}^{[i]})$. Minimizing the objective function is an instance of weighted ridge regression.

IV. Experiments

To evaluate the LaDiPS algorithm, we compare it to HiREPS on a planar reaching task and a simulated table tennis task. For each task, we first explain how the task is set up and what are the hyper parameters for the learning algorithm. Second, we compare the performance of the two algorithms as well as examine the diversity of the found solutions. We want to show that LaDiPS can learn a diverse set of solutions and learn high quality solutions.

A. Experiment Setup Reaching Point Task

The experiment is set up such that the learning algorithm can find multiple sub-policies to solve the task. The task we use is a planar reaching task. The goal is to move the end effector of a three joint, three link robot arm to certain positions at certain time points. The first time point is at time step 50 and the second time point is at time step 100. They can be seen in Fig. 2 as blue and red dots respectively. At each of these times the robot has to reach one of the possible reaching points. Fig. 2 shows one learned solutions. After the movement is executed, a score is given based on the squared distance of the end effector to the closest reaching point at the specified time.

To generate a trajectory, we use Dynamic Movement Primitives (DMP) [9]. We use one DMP for each joint. The DMPs generate the desired angles of the joints. The goal position of the DMP is set such that the arm reaches to the red point. Each DMP uses five basis functions. The linear weights of these five basis functions compose the parameter vector $\boldsymbol{\theta}$. To track the generated trajectory, we use a PD controller with a feedforward term. The initial context $\boldsymbol{x}$ is the angle at the base of the robot. It is uniformly distributed between -0.5 and $+0.5$ radians. The other two angles are set to zero. If all angles are zero, then the robot stands up straight and touches the red reaching point.

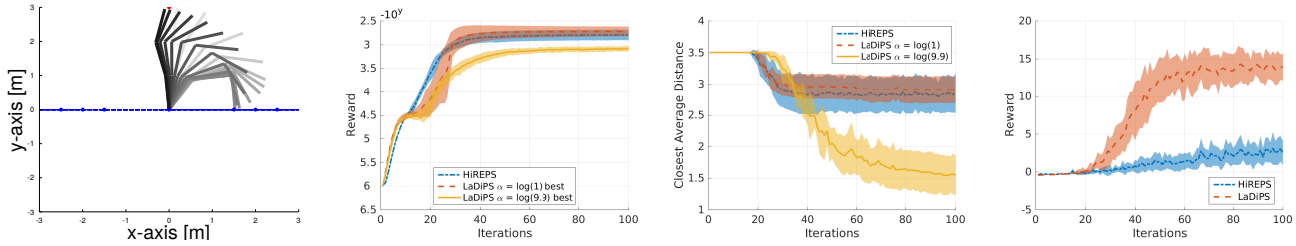

Fig. 2: The left plot is an illustration of a successful trajectory of the three link, three joint robot arm. The different shades of grey show the different positions of the robot over time. The center left plot shows the average reward for the best sub-policy of HiREPS and LaDiPS on the reaching point task. The center right plot shows how far the sub-policies are away from the reaching points. The right plot shows the average reward on the table tennis task.

For the experiments, 20 trials are evaluated with each trial consisting of 100 iterations. In each iteration, 500 samples (rollouts) are generated. For LaDiPS, we set ξ, ε and the reduction h of the parameter β to 1, 0.5 and 2 respectively. The parameter α is set to $\ln(1)$ to achieve one sub-policy with very high reward and we test another setting with α set to $\ln(9.9)$ to achieve high diversity. For HiREPS, the parameter ε is set to 0.5 and κ is set to 98% of the entropy of the previous iteration [7]. These parameter gave the best performance for HiREPS. The mixture model used for both learners is a mixture model with softmax gating and 10 Gaussian sub-policies. The features of the Gaussian sub-policies are linear and for the gating network the features are quadratic.

B. Results Reaching Point Task

First, we compare the quality of the found sub-policies. The center left plot of Fig. 2 shows the average reward over all trials of the best sub-policy[1] of HiREPS and LaDiPS. The plot shows that the best sub-policy of LaDiPS for $\alpha = \ln(1)$ is slightly better than the best sub-policy of HiREPS, whereas the best sub-policy for $\alpha = \ln(9.9)$ is worse.

Next, we compare the diversity of the found sub-policies. The diversity is evaluated by generating 10 samples from each sub-policy and computing for each reaching point the average distance to the samples from each sub-policy. We take for each reaching point the smallest average distance to a sub-policy. We cap this distance at 0.25 (half the distance between two close reaching points) and sum these distances over all reaching points. This distance measure represents diversity well as it is small only if many reaching points have one sub-policy that reliably comes close to each of these reaching points. The distance is plotted in the center right plot of Fig. 2. We can see that for $\alpha = \ln(1)$ LaDiPS is slightly less diverse than HiREPS. However, LaDiPS with $\alpha = \ln(9.9)$ is much more diverse than the other two.

C. Experiment Setup Table Tennis Task

The goal of the simulated table tennis task is to return a ball using a robot arm with a racket as its end effector. The arm itself has 6 degrees of freedom and is attached to a base that can move in all 3 dimensions. To generate a trajectory for the robot we use 9 Dynamic Movement Primitives (DMP) [9] one for each degree of freedom. The ball is shot from the center of the opponents side to the robot. The point to which the ball is shot varies by ± 42.5cm on the x-axis[2] around the initial end effector position. On the y-axis[2] and z-axis[2] the ball only varies around the initial end effector position by ± 0.01cm. The context $\boldsymbol{x}$ is the initial velocity of the ball when it is shot from the opponents side. The reward is given based on how close the ball is played to one of the opponents corners or how close to the opponents net.

The parameters of the policy are the goal position of the DMP. We use a mixture model with 8 Gaussians as sub-policies and a softmax as gating network. The features are the same as in the Reaching Point Task. The weights of the DMP and the initial mean of the Gaussian sub-policies are set by imitation learning from one forehand and one backhand trajectory. The imitation learning trajectories are shown in the additional video. The DMP weights are kept fixed after the imitation learning. Half the sub-policies use the DMP weights learned from the backhand stroke and the other half use the DMP weights learned from the forehand stroke.

For the experiments, 10 trials are run for each algorithm with each trial consisting of 100 iterations. In each iteration 50 samples (rollouts) are generated, however, we use importance weighting to reuse up to 1000 samples. The parameters used for learning LaDiPS are 0.01 for ε and 0.5 for ξ. For the parameter β, the reduction h was 3 and α was set to $\ln(3)$. For HiREPS, the parameter ε is set to 0.5 and κ is set to 98% of the entropy of the previous iteration. These parameter gave the best performance for HiREPS.

D. Results Table Tennis Task

For the table tennis task, we are interested if we can find diverse solutions. To examine the diversity of the solutions, we compare the single backhand stroke used for the initial imitation learning with two backhand strokes that were learned by LaDiPS (Fig. 3). We can see that for the initial backhand stroke mostly the arm is moved while the base only moves slightly. For the learned trajectories, the base moves a lot more. The first learned stroke (Fig. 3 middle row) uses the base movement to catch a ball played to the forehand

[1]The plot for the best sub-policy shows the average reward of the sub-policy that has the best reward in the *last* iteration.

[2]From the players perspective standing in front of the table the x-axis is left and right, the y-axis is forward and backward and the z-axis is up and down

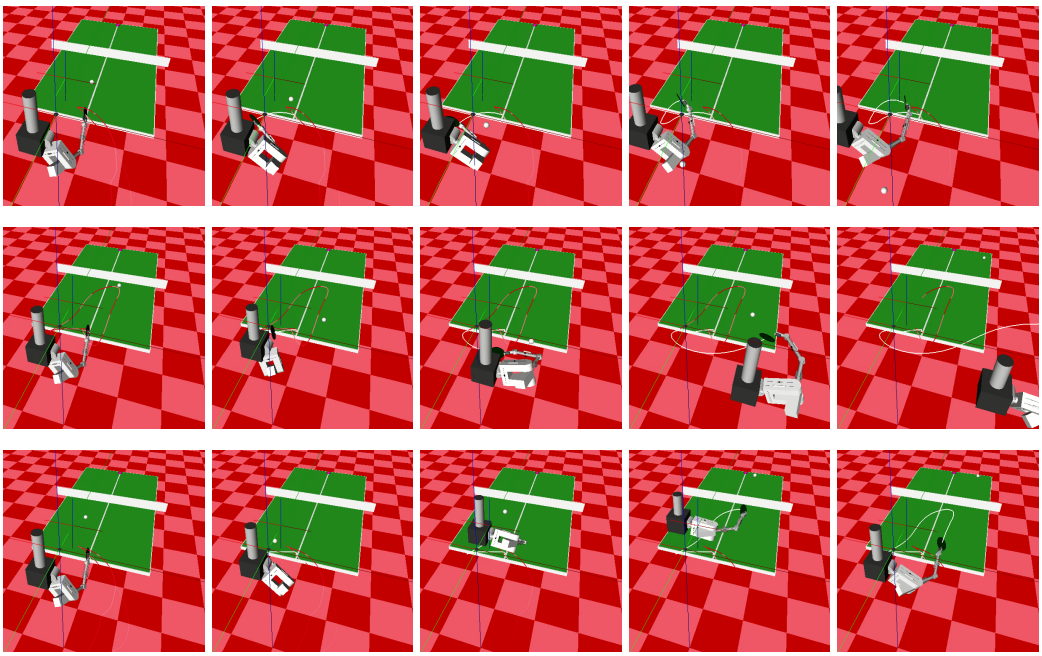

Fig. 3: Each row shows a backhand stroke. The first row is the single stroke used in the initial imitation learning and the second and third rows are learned strokes.

side far outside the range of the backhand trajectory for imitation learning. The second learned stroke (Fig. 3 bottom row) moves the base towards the ball using the momentum of the base as part of the swing. The additional video shows three different learned solution where each solution hits the ball to one of the three high reward point (the two far corners and the center close to the net). From these examples, we see that LaDiPS can learn diverse solutions even on complex task like this table tennis simulation. In terms of quality of the policy, we see in Fig. 2 that LaDiPS outperforms HiREPS.

V. Conclusion

In this paper, we proposed a novel hierarchical reinforcement learning framework that learns on both layers of a hierarchical policy that is composed of a gating policy and several sub-policies.

Our new algorithm optimizes a mixture model using individual learning processes for the gating and the sub-policies. Both layers are connected by regression problems that are used to estimate the corresponding reward models. While we implemented learning algorithms that are based on the Model-Based Relative Entropy Stochastic Search (MORE) [14] algorithm for the individual layers, the whole framework is more general and other algorithms could be used that allow for the use of sample reweighting with importance weights.

In the experiments, we showed that LaDiPS can control the diversity of the policy to find more solutions than other Hierarchical Reinforcement Learning algorithms. Additionally, LaDiPS could find diverse solutions on complex task such as the simulated table tennis task.

In the future, we want to add more layers to the LaDiPS framework and experiment with different algorithms on the different layers. Moreover, we will apply our algorithms to more complex tasks and real robot applications.

References

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *IJRR*, p. 0278364913495721, 2013.

[2] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[3] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[4] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *IJRR*, vol. 32, no. 3, pp. 263–279, 2013.

[5] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *JAIR*, vol. 13, pp. 227–303, 2000.

[6] M. Ghavamzadeh and S. Mahadevan, "Hierarchical policy gradient algorithms," in *ICML*, 2003, pp. 226–233.

[7] C. Daniel, G. Neumann, and J. R. Peters, "Hierarchical relative entropy policy search," in *AISTATS*, 2012, pp. 273–281.

[8] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics." *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[9] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines*. Springer, 2006, pp. 261–280.

[10] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," *NIPS*, pp. 1043–1049, 1998.

[11] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[12] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search," in *AAAI*. Atlanta, 2010.

[13] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *NIPS*, 2014, pp. 1071–1079.

[14] A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. P. Reis, and G. Neumann, "Model-based relative entropy stochastic search," in *NIPS*, 2015, pp. 3523–3531.

[15] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.