# Learning versatile solutions with policy search

**Vielfältigen Lösungen lernen mit Policy Search**
Bachelor-Thesis von Felix End aus Bad Homburg
Tag der Einreichung:

1. Gutachten: Prof. Dr. Gerhard Neumann
2. Gutachten: Prof. Dr. Jan Peters
3. Gutachten: Dr. Riad Akrour

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Learning versatile solutions with policy search
Vielfältigen Lösungen lernen mit Policy Search

Vorgelegte Bachelor-Thesis von Felix End aus Bad Homburg

1. Gutachten: Prof. Dr. Gerhard Neumann
2. Gutachten: Prof. Dr. Jan Peters
3. Gutachten: Dr. Riad Akrour

Tag der Einreichung:

# Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 30. September 2015

(Felix End)

# Abstract

Damage or wear of robot parts or changes in the environment can cause the previously optimal behavior of a robot to become inadequate. Having backup solutions in such a situation can allow the robot to continue doing its task. The majority of Reinforcement Learning (RL) algorithms is not designed to find backup solutions. This thesis presents a new hierarchical RL algorithm that builds on the ideas of the Hierarchical Relative Entropy Policy Search (HiREPS) algorithm to learn multiple solutions. The policy that is learned is a 'mixed options' policy, which consists of a gating policy and option policies. The gating policy selects the option policy, which then chooses an action. The different option policies represent different solutions.
The presented algorithm mimics the two level structure of the 'mixed options' policy. It has an upper level which optimizes the gating and a lower level which optimizes the option policies. Both the upper and the lower level optimization are extensions of Relative Entropy Policy Search (REPS). Subsequently, the whole algorithm belongs to the group of policy search algorithms.
The experiments show that the new algorithm finds more versatile solutions than HiREPS and its best solution is in quality similar to that of HiREPS.

# Zusammenfassung

Wenn sich das Verhalten eines Roboters durch Beschädigung oder Abnutzung ändert oder wenn sich die Umgebung des Roboters ändert, können Bewegungen, die zuvor optimal waren, auf einmal unzureichend sein. In dieser Situation wäre es nützlich Ersatzlösungen zu haben, die dem Roboter erlauben weiter zu arbeiten. Die meisten Algorithmen für Reinforcement Learning (RL) sind nicht dafür konzipiert, Ersatzlösungen zu lernen. In dieser Bachelorarbeit wird ein neuer Algorithmus für das Lernen von unterschiedlichen Lösungen für hierarchische RL Probleme vorgestellt. Dieser Algorithmus basiert auf dem Hierarchical Relative Entropy Policy Search (HiREPS) Algorithmus und wie bei HiREPS wird eine 'mixed Options' Policy gelernt. Diese Policy besteht aus einer Gating Policy und vielen Option Policies, welche unsere verschiedenen Lösungen darstellen. Die Aufgabe der Gating Policy ist es, eine Option Policy auszuwählen, welche dann eine Aktion ausführt.
Der vorgestellte Algorithmus spiegelt die zweistufige Hierarchie der 'mixed Options' Policy wider. Auf der höheren Stufe wird die Gating Policy optimiert und auf der unteren Stufe werden die Option Policies optimiert. Beide Stufen der Hierarchie sind jeweils Erweiterungen von Relative Entropy Policy Search (REPS) und dementsprechend gehört der vorgestellte Algorithmus zur Gruppe der Policy Search Algorithmen.
Die Experimente zeigen, dass der vorgestellte Algorithmus vielseitigere Lösungen als HiREPS findet und die beste Lösung, die er findet, erziehlt ähnlich hohen Reward wie die beste Lösung die HiREPS findet.

# Acknowledgments

First, I want to thank my supervisors Gerhard Neumann and Riad Akrour for their advice and guidance throughout this thesis. I have been tremendously impressed by and grateful for their patience with me whenever I got stuck.
I would also like to thank my family and friends for their continuous support.
Lastly, I would also like to thank Thomas Hesse and Kevin Luck for providing the template for this thesis.

# Contents

# Figures

## List of Figures

# Abbreviations, Symbols and Operators

## List of Abbreviations

| Notation | Description |
|---|---|
| DMP | Dynamic Movement Primitives |
| EM | Expectation–Maximization |
| HiREPS | Hierarchical Relative Entropy Policy Search |
| IMRL | Intrinsically Motivated Reinforcement Learning |
| KL-divergence | Kullback-Leibler divergence |
| LHiREPS | Layered Hierarchical Relative Entropy Policy Search |
| MDP | Markov Decision Process |
| REPS | Relative Entropy Policy Search |
| RL | Reinforcement Learning |

## List of Symbols

| Notation | Description |
|---|---|
| $o_j$ | The option whose option policy we are currently optimizing is denoted with the index $j$. This notation is only relevant for the option optimization of our new approach. |
| $(\cdot)^{[i]}$ | We use the letter $i$ in square brackets to index the samples from a distribution. |
| $\pi(\cdot)$ | The policy that is optimized is denoted with $\pi$. In this thesis, all policies are parameterized, but for brevity we write $\pi(a\|s)$ instead of $\pi(a\|s;\theta)$. |
| $\mu(s)$ | The state distribution. Very similar is also $\mu_o(s)$ which is the state distribution for an option, i.e., $p(s\|o)$. All state distributions we consider here are initial state distributions. |
| $p(o\|s,a)$ | We call this probability ”responsibility of an option” or just ”responsibility”. |
| $\tilde{p}(o\|s,a)$ | The variable $\tilde{p}(o\|s,a)$ is the estimated value of $p(o\|s,a)$ at the beginning of the optimization. It is fixed during the optimization as part of an EM-procedure explained in the HiREPS section. |
| $\hat{p}(o\|s,a)$ | The variable $\hat{p}(o\|s,a)$ is used instead of $p(o\|s,a)$ as we cannot solve for $p(o\|s,a)$. It is different from $\tilde{p}(o\|s,a)$ because it is continually updated during the optimization. |
| $q(\cdot)$ | We use the letter $q$ to denote the old probabilities from which we sampled the sates or actions. |
| $p(\cdot)$ | Probabilities for which none of the special notation above applies are denoted with $p$. |

## List of Operators

| Notation | Description | Operator |
|---|---|---|
| $\mathrm{H}_\mathrm{p}$ | The entropy with respect to the distribution $p$ | $\mathrm{H}_p[\,\cdot\,]$ |
| $D_{\mathrm{KL}}$ | The Kullback-Leibler divergence | $D_{\mathrm{KL}}(\,\cdot\,\|\|\,\cdot\,)$ |

# 1 Introduction

In today's world, many different kinds of robots exist. From household cleaning robots, to manufacturing robots, to a few humanoid robots, these robots come in different complexities. While for the simpler of these robots programming them directly can work well, for more complex robots using machine learning techniques to learn the desired behavior can be more effective. Learning a behavior with machine learning takes some time and generally requires human supervision. However, after the behavior is learned, the robot can perform the task forever without further learning, at least in theory. Robots that do not work in a controlled environment, like working with humans, can accidentally get damaged and even robots in controlled environments will wear out over time. The damaged or worn out parts may change the dynamics of the robot. Additionally, robots may be used in environments slightly different then the ones they were trained for. As an example a robot may be moved to a smaller room that requires a more concise behavior than it was trained for. In all of these examples, the robot could still be able to perform the desired task with the right behavior, but the behavior that it learned may not work anymore. In this case the only option would be to revert the change, if that is possible, or retrain the robot. However, retraining could be avoided if alternative solutions for the task had been learned earlier. For example, for a robot with a damaged or worn out part an alternative solution which minimizes the use of this part may still work good enough. Having such backup solutions may be especially useful if retraining or replacing the robot is very expensive or not possible in time, for instance, in case of an emergency.

However, most machine learning algorithms are designed to learn only one optimal solution to a problem and cannot learn versatile solutions. One of the algorithms that can learn versatile solutions is HiREPS [1]. Unfortunately, it does not reliably find versatile solutions. It will often find few or no alternative solutions. This effect is not so surprising as the main goal of HiREPS is not to find backup solutions but to find better solutions by exploiting the hierarchical structure inherent in many problems. Finding backup solutions is only a secondary interest of HiREPS.

In this thesis, we present a new algorithm to learn versatile solutions more reliably than HiREPS. We call this algorithm Layered Hierarchical Relative Entropy Policy Search (LHiREPS) because it is an extension of HiREPS and structured in a two layer hierarchy. The HiREPS algorithm is itself an extension of the REPS algorithm, which is a state of the art policy search algorithm [2]. HiREPS extends REPS by using a 'mixed options' policy to learn different solutions. This 'mixed options' policy consists of option policies that represent different solutions and a gating policy that chooses one of these options based on the state of the environment. HiREPS learns this policy by optimizing the whole 'mixed options' policy in one optimization procedure.
The main idea of our new algorithm LHiREPS is also to use a 'mixed options' policy, but to optimize the gating policy and the options policies individually. This partition allows us to enforce specific constraints for the gating or options policy in the respective optimization steps more easily. One new constraint we enforce in the gating optimization bounds the entropy of the distributions over options given the state. This bound is used to regulate the number of options that have non-zero probability. This extension is important because HiREPS does not learn backup solutions reliably due to the fact that the probability of all options except the best one often drop to zero. If this unbalanced assignment happens, no more samples are drawn for these options and they stop being updated. For this reason, we add the constraint to better control the entropy of the option distribution, which results in a more versatile solution space. To optimize the gating policy, we estimate a new reward $R_{so}$. With this reward the gating policy is optimized in a similar manner as REPS optimizes its policy. Although, the structure of the reward as well as the structure of our gating policy allow us to slightly simplify the constraint optimization problem and to directly update our policy.
The option policy optimization problem is very similar to the one of HiREPS, sharing all of its constraints in a slightly altered form. However, we also explore adding additional constraints to avoid the EM-like procedure that HiREPS does.

In the next chapter of this thesis, we explain what kind of tasks LHiREPS is designed for. In the same chapter, we will also explain REPS and HiREPS in more detail. In the third chapter, we look at some similar work to ours. The fourth chapter explains LHiREPS and in the fifth chapter we examine some experimental results. Lastly, the sixth chapter will conclude this thesis with a summary of the main accomplishments of this work and a short look on possible future work. The details of the derivation of LHiREPS are left out of the main part of the document, but they can be found in the appendix.

# 2 Foundation

In this chapter, we establish the background for our new approach. Our approach is part of the field of machine learning. More specifically it is designed for Reinforcement Learning with policy search. In the first sections, we will introduce the field of Reinforcement Learning and its sub-field policy search. In the second and third section, we introduce the two algorithms REPS and HiREPS, which are policy search algorithms and, more importantly, they are the basis of our new algorithm.

## 2.1 Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning that models the trial-and-error learning we are familiar with in our everyday lives [3]. In the following we will explain the general task setup, how to update policies in the sub-field of policy search and how the whole learning process for RL works.

### 2.1.1 Task Setup

The RL tasks we are considering can be formalized as Markov Decision Process (MDP). A MDP involves an agent and an environment. The agent interacts with the environment for every time step of a sequence of discrete time steps ($t = 0, 1, 2, 3, \dots$). At each time step $t$ the agent chooses an actions $a_t \in \mathcal{A}(s_t)$ based on the state $s_t \in S$ of the environment. Here the set $\mathcal{A}(s_t)$ is the set of all possible actions for state $s_t$ and the set $S$ is the set of all states. When the agent chooses an action $a_t$ the environment changes from state $s_t$ to the state $s_{t+1}$ based on the transition probability $p(s_{t+1}|s_t, a_t)$. Additionally, the agent receives a reward $r_{t+1} \in \mathcal{R}$ at the next time step based on the the action chosen and the state of the environment. We will only consider the episodic case were the interactions end at some time step $T$. In the episodic case the goal is to maximize the expected return. The return is given by

$$R = \sum_{k=t}^{T} r_k$$

To improve the agent a learning algorithm is used. The learning algorithms we present in this thesis are formulated so that they are only for MDPs with one step, i.e., $T = 1$. In this case the return is equal to the first reward, so it only depends on the initial state and the corresponding action. We denote this return as $R_{sa}$ where $s$ is the initial state and $a$ is the first action. We will generally refer to $R_{sa}$ as "reward" as the return is equal to the only reward we receive.
Nonetheless, we can also indirectly use these learning algorithms on MDPs that require multiple steps by using a function that can generate multiple actions based on one input parameter. From the learning algorithms point of view the original MDP with multiple steps combined with the function is a new MDP with only one step. The action the agent produces for the new MDP is the input parameter of the function, which generates the actions for the original MDP. The reward of this new MDP is the return of the original MDP.

### 2.1.2 Policy Search and Policies

Policy search is a sub-field of RL [4]. In policy search the actions the agent produces are given by a policy. This policy is a parameterized function $a = f_\theta(s)$ or probability $p(a|s;\theta)$. It is improved by the learning algorithm by updating the parameter $\theta$. An example of a parameterized policy is a Gaussian policy that is linear in features. It has the form

$$p(a|s;\theta) = \mathcal{N}(a|b + A\phi(s), \Sigma) \qquad \text{with } \theta = (b, A, \Sigma).$$

The function $\phi$ is our feature vector which we use to transform our state into a more useful space. The parameter $\theta$ is changed to improve the policy.

Another example of a policy is a 'mixed options' policy. It consists of a gating policy $p(o|s)$ and many option policies $p(a|s,o)$. The gating policy generates a discrete option $o$ which determines which option policy generates the action. An example of an option policy would be a Gaussian policy that is linear in features. Each of these option policy can

represent a different solution. These many solutions are more versatile than the one solution a single Gaussian policy can represent. For instance, one option policy could be a forehand stroke and another one could be a backhand stroke for robot table tennis.

An example of a gating policy would be a softmax distribution, i.e.,

$$p(o|s) = \frac{\exp\left(\theta_o^T \phi(s)\right)}{\sum_o \exp\left(\theta_o^T \phi(s)\right)}.$$

The parameters $\theta_o$ are altered to change the probability for the different options and $\phi$ is again the feature vector. Note that we wrote $p(o|s)$ instead of $p(o|s;\theta)$. We leave out $\theta$ to keep the notation simpler as all policies we will encounter will be parameterized, so there is no need to distinguish between unparameterized or parameterized policies.

This kind of mixture model, with softmax gating and Gaussians linear in features as option policies, is what we will be using for our experiments and the softmax gating will even play an important part in our new algorithm.

### 2.1.3 Learning Process

The RL process is generally iterative. For our one step MDP, we sample many initial states and generate for each state an action from our policy independently. For each state-action pair, we obtain a reward. These samples of rewards and state-action pairs are then used to update the policy with the learning algorithm. After the policy update, the process repeats. This repetition is necessary because the initial policy will generally be bad. The best samples it obtains will be far off from the (yet unknown) sample with the highest reward, so changing it to perfectly match the best sample it currently knows is a bad idea. Instead, we change the policy only slightly to explore more around the most promising samples in the hope that this exploration leads us to the very best sample. Generally, we can only guarantee to find a local optimum of the reward function but not a global optimum.

Connected to this iterative approach is the following commonly used notation. The policy we want to improve is denoted as $\pi$. For instance, the Gaussian policy $p(a|s)$ from the previous example, will be written as $\pi(a|s)$. The state distribution $p(s)$ is denoted by $\mu(s)$. The old distribution, we sampled from, is denoted with $q$. Any other probability is denoted with the letter $p$.

## 2.2 Relative Entropy Policy Search

Episode-based contextual Relative Entropy Policy Search (in the following just REPS) [2] is an algorithm that can learn good policies for the kind of RL problem we described in the previous section. Further, REPS is also the basis of LHiREPS we will introduce later.

The goal of the REPS algorithm is to maximize the expected reward with respect to $\pi(a|s)$. This maximization is not easy as it requires that we have many action samples for each state [4]. To deal with this issue, REPS optimizes the expected reward for $p(a,s) = \pi(a|s)\mu(s)$ instead, i.e,

$$\max_{\pi(a|s)\mu(s)} \iint_{a,s} \pi(a|s)\mu(s)R_{sa} \, \mathrm{d}a \, \mathrm{d}s.$$

The solution $p(a,s)$ of this maximization has to "respect" $\mu(s)$ in the sense that $\mu(s) = \int_a p(a,s) \, \mathrm{d}a$ has to hold for all states $s$. However, if $s$ is continuous we have an infinite number of constraints, which makes the optimization problem intractable [5]. This issue is solved in the REPS algorithm by requiring instead that the observed average feature vector matches the expected feature vector

$$\hat{\phi} = \iint_{a,s} \pi(a|s)\mu(s)\phi(s) \, \mathrm{d}a \, \mathrm{d}s.$$

Optimizing the constraint optimization problem we currently have would cause the algorithm to converge to the best sample in the first iteration and it would not explore anymore in any future iterations. This premature convergence can be avoided -as explained in the previous section- by only allowing small changes in the distribution of the policy in each

update. To enforce small changes, a notion of distance between the new and the old policy is needed. REPS uses the Kullback-Leibler divergence (KL-divergence)[1] to quantify how much one distribution changed from the other.
The KL-divergence between a distribution $p(x)$ and a distribution $q(x)$ is defined as

$$D_{\mathrm{KL}}(p(x)||q(x)) = \int_x p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x.$$

REPS controls how much the policy can change in each iteration by setting an upper bound on the average KL-divergence between the new and the old joint distribution.

Lastly, it is also requires that $p(a,s)$ is a proper distribution that means it has to integrate to 1.
Combining theses three constraints yields the REPS optimization problem

$$\begin{aligned}
&\max_{\pi(a|s)\mu(s)} \iint_{a,s} \pi(a|s)\mu(s)R_{sa} \,\mathrm{d}a\,\mathrm{d}s,\\
&\text{s.t:}\quad \varepsilon \geq D_{\mathrm{KL}}(\pi(a|s)\mu(s)||q(a,s)),\\
&\qquad \hat{\phi} = \iint_{a,s} \pi(a|s)\mu(s)\phi(s) \,\mathrm{d}a\,\mathrm{d}s,\\
&\qquad 1 = \iint_{a,s} \pi(a|s)\mu(s) \,\mathrm{d}a\,\mathrm{d}s.
\end{aligned}$$

It can be solved using the method of Lagrangian multipliers [6], which gives a closed form solution for $p(a,s)$

$$p(a,s) \propto q(a,s) \exp\left(\frac{R_{sa} - \theta^T \phi(s)}{\eta}\right).$$

The parameters $\eta$ and $\theta$ are the Lagrangian multipliers. The parameter $\theta$ corresponds to the $\phi$-constraint. It creates a baseline $\theta^T\phi(s)$ which offsets the reward $R_{sa}$ depending on the state. The parameter $\eta$ corresponds to the KL-divergence-constraint. It guarantees that $p(a,s)$ does not differ too much from $q(a,s)$ by scaling the offset reward appropriately.
The Lagrangian multipliers are obtained by minimizing the dual function for $\eta > 0$. The dual function is given by

$$g(\eta,\theta) = \eta \log\left(\iint_{a,s} q(a,s) \exp\left(\frac{R_{sa} - \theta^T\phi(s)}{\eta}\right) \mathrm{d}a\,\mathrm{d}s\right) + \eta\varepsilon + \theta^T\hat{\phi}.$$

After optimizing the dual function, we have the optimal probability $p(a,s)$ for our samples. To generalize this probability to all samples, we change the parametric model of our policy (e.g., our Gaussian linear in features) to be close to these optimal probabilities. This policy update is done by minimizing the KL-divergence between the optimal probabilities and the parametric model times the state distribution. This minimization can simplified by

$$\begin{aligned}
&\min_{\pi_m(a|s)} D_{\mathrm{KL}}(p(a,s)||\pi_m(a|s)\mu(s))\\
&\quad = \min_{\pi_m(a|s)} \iint_{a,s} p(a,s) \log\left(\frac{p(a,s)}{\pi_m(a|s)\mu(s)}\right) \mathrm{d}a\,\mathrm{d}s\\
&\quad = \min_{\pi_m(a|s)} \iint_{a,s} -p(a,s)\log\pi_m(a|s) + C(a,s) \,\mathrm{d}a\,\mathrm{d}s\\
&\quad \approx \max_{\pi_m(a|s)} \frac{1}{N} \sum_{i=1}^{N} \frac{p(a^{[i]},s^{[i]})}{q(a^{[i]},s^{[i]})} \log\pi_m(a^{[i]}|s^{[i]})
\end{aligned}$$

with $C(a,s) = \log\left(\frac{p(a,s)}{\mu(s)}\right)$. The parametric model of our policy is denoted by $\pi_m(a|s)$. The second equation can be understood as an expectation, which is why it can be approximated by taking the average over the samples. The index $(\cdot)^{[i]}$ denotes the $i$-th sample. The number $N$ is the total number of samples.
After simplifying, the minimization of the KL-divergence turns out to be equivalent to updating our policy with weighted maximum likelihood.

[1] The Kullback-Leibler divergence is not a distance metric because it is not symmetric.

## 2.3 Hierarchical Relative Entropy Policy Search

The HiREPS [1] algorithm is an extension of REPS. As with REPS, we only look at HiREPS for episodic problems. The main idea of HiREPS is to use a hierarchical policy to deal with problems that non-hierarchical policies struggle with, for instance, averaging over multiple local optima. The hierarchical policy we consider here is the 'mixed options' policy introduced in the 'Reinforcement Learning and Policy Search' section.

HiREPS learns this policy by using a similar optimization problem as REPS but, instead of optimizing for the joint of just the action and states $p(a,s) = \pi(a|s)\mu(s)$, it also optimizes for the options $p(a,s,o) = \pi(a,o|s)\mu(s)$.

Furthermore, HiREPS also adds a new constraint to make the different options diversify. This constraint is important because learning two identical options (i.e., both produce the same kind of actions) is useless.

To force diverse options, HiREPS uses the entropy of the responsibilities $p(o|s,a)$ because the entropy of a distribution is higher if the distribution is closer to being uniformly distributed and lower if the distribution is more spiked. For the responsibilities $p(o|s,a)$ being more uniformly distributed for a state-actions pair $(a,s)$ means many of our options could have produced the action $a$ for that state $s$. On the other hand, if the responsibility distribution is very spiked, then only those options for which $p(o|s,a)$ spiked are likely to have produced the action $a$ in the state $s$. All other options are unlikely to have produced $a$ for $s$ which means the options are diverse for $a$ and $s$. Bringing these ideas together we see that a lower entropy for the responsibilities means more diverse solutions. HiREPS uses exactly this property to enforce diversity with the constraint

$$\kappa \geq \iint_{a,s} p(a,s)\mathrm{H}_p[o|s,a]\,\mathrm{d}a\,\mathrm{d}s \qquad \text{with} \quad \mathrm{H}_p[X] = -\sum_x p(x)\log p(x).$$

The operator $\mathrm{H}_p[\,\cdot\,]$ is the entropy with respect to the probability $p$.

This constraint together with the others yields the following optimization problem (which is not the final HiREPS optimization problem):

$$\begin{aligned}
&\max_{\pi(a,o|s)\mu(s)} \iint_{a,s} \sum_o \pi(a,o|s)\mu(s)R_{sa}\,\mathrm{d}a\,\mathrm{d}s,\\
&\text{s.t:}\quad \varepsilon \geq \iint_{a,s} \sum_o \pi(a,o|s)\mu(s)\log\frac{p(a,s)}{q(a,s)}\,\mathrm{d}a\,\mathrm{d}s,\\
&\qquad \kappa \geq -\iint_{a,s} p(a,s)\sum_o p(o|s,a),\log p(o|s,a)\,\mathrm{d}a\,\mathrm{d}s,\\
&\qquad \hat{\phi} = \iint_{a,s} \sum_o \pi(a,o|s)\mu(s)\phi(s)\,\mathrm{d}a\,\mathrm{d}s,\\
&\qquad 1 = \iint_{a,s} \sum_o \pi(a,o|s)\mu(s)\,\mathrm{d}a\,\mathrm{d}s.
\end{aligned}$$

Unfortunately, the marginal $p(a,s)$ and the responsibilities $p(o|s,a)$ inside the logarithms do not allow for a closed form solution. HiREPS solves this issue by expressing the marginal in terms of responsibilities and joint distribution $p(a,s) = p(a,s,o)/p(o|s,a)$ and by keeping the responsibilities inside the logarithms fixed. Fixing the responsibilities works because it creates a lower bound on the original optimization problem. The lower bound means that we get an algorithm closely related to the Expectation-Maximization (EM) algorithm. The E-step is the estimation of the responsibilities and the M-step consists of solving the optimization problem with fixed responsibilities.

The HiREPS optimization problem with fixed responsibilities is given by

$$
\begin{aligned}
&\max_{\pi(a,o|s)\mu(s)} \iint_{a,s} \sum_{o} \pi(a,o|s)\mu(s)R_{sa} \,\mathrm{d}a\,\mathrm{d}s, \\
&\text{s.t:}\quad \varepsilon \geq \iint_{a,s} \sum_{o} \pi(a,o|s)\mu(s) \log \frac{\pi(a,o|s)\mu(s)}{q(a,s)\tilde{p}(o|s,a)} \,\mathrm{d}a\,\mathrm{d}s, \\
&\qquad \kappa \geq \iint_{a,s} p(a,s) \sum_{o} p(o|s,a) \log \tilde{p}(o|s,a) \,\mathrm{d}a\,\mathrm{d}s, \\
&\qquad \hat{\phi} = \iint_{a,s} \sum_{o} \pi(a,o|s)\mu(s)\phi(s) \,\mathrm{d}a\,\mathrm{d}s, \\
&\qquad 1 = \iint_{a,s} \sum_{o} \pi(a,o|s)\mu(s) \,\mathrm{d}a\,\mathrm{d}s.
\end{aligned}
$$

The responsibilities with a tilde $\tilde{p}(o|s,a)$ are the ones we keep fixed during the optimization.
Solving this optimization problem with the method of Lagrangian multipliers yields a closed form solution for $p(a,s,o)$

$$
p(a,s,o) \propto q(a,s)\tilde{p}(o|s,a)^{1+\xi/\eta} \exp\left(\frac{R_{sa} - \theta^T \phi(s)}{\eta}\right).
$$

The parameters $\eta$, $\theta$ and $\xi$ are the Lagrangian multipliers. They are obtained by minimizing the dual function

$$
g(\eta,\theta,\xi) = \varepsilon\eta + \kappa\xi + \theta^T\phi(s) + \eta \log\left(\iint_{a,s} q(a,s) \sum_{o} \tilde{p}(o|s,a)^{1+\xi/\eta} \exp\left(\frac{R_{sa} - \theta^T\phi(s)}{\eta}\right) \,\mathrm{d}a\,\mathrm{d}s\right)
$$

for $\eta > 0$ and $\xi > 0$.

After optimizing the dual function, the joint probabilities $p(a,s,o)$ are used in a similar manner as in REPS to update the parametric model of the policy by minimizing $D_{\mathrm{KL}}(p(a,s,o)||\pi_m(a,o|s)\mu(s))$ for $\pi_m(a,o|s)$. The 'mixed options' policy is denoted by $\pi_m(a,o|s)$.

# 3 Related Work

The idea of hierarchical RL is not new and other authors have developed algorithm and frameworks for hierarchical RL [7, 8, 9, 10]. In this section we want to go over some of these and see how LHiREPS fits in.
We start by looking at the motion templates framework [7]. This framework structures the policies for RL in form of a two level hierarchy. The lower level policies are called motion templates. They correspond to our concept of ”options”. The upper level policies are selection policies that choose which motion template is executed and with which parameters it is executed. What sets this framework apart from other frameworks -like the option framework [8], MAXQ framework [10] or HAM [9]- is that it is designed for learning the motion templates as well as the selection policies from the same samples. Our approach fits very well into this framework as it has the proposed two level hierarchy and it is an algorithm that learns the policies of both levels with the same samples.

Besides the motion template framework, there have been many other frameworks for hierarchical RL. One of the most well known ones is the options framework by Sutton et al. [8]. Unlike the motion template framework, it considers arbitrary hierarchies. The policies in the options framework are probabilities over abstract options or concrete actions where only the actions are actually executed in the environment. The abstract options are policies themselves which will execute either other options or actions. This hierarchy is more flexible than our two level hierarchy, but in the framework the options are only learned with the help of subgoals. This approach is different from our algorithm, which learns options without the help of subgoals.
Regarding how to learn these options with subgoals, there is an interesting extension of the options framework by Stolle et al. [11] that answers this question. It learns options by randomly generating subtasks of the form ’starting in state $s$ find a way to change the state of the environment to state $t$’. For these randomly generated tasks, it learns how to best get from $s$ to $t$. The states that were visited most often are probably important and become target states for options. For all target states, the algorithm learns a policy to reach these target states. These policies are the new options. What is interesting about this idea is that the options are learned without the task that they will be used for. This concept of learning without the actual task is similar to the ideas of Intrinsically Motivated Reinforcement Learning (IMRL) [12]. In IMRL the agent generates internal rewards called ’intrinsic rewards’ whose purpose it is to promote interesting actions. These intrinsic rewards are used to learn useful options -called skills in IMRL- that may help solve the task at hand and future tasks faster.
In both of these approaches, a goal of learning options is to use them for unknown future tasks. This goal is very similar to our goal of finding options as backup solutions for (unknown) future tasks in which the goal is still the same but the environment has changed. However, our approach only uses rewards from the environment and does not generate rewards for interesting actions itself. Since our algorithm does not use intrinsic rewards, it is not an IMRL algorithm, even though its ideas are related to IMRL.

# 4 Layered Hierarchical Relative Entropy Policy Search

## 4.1 Overview

In this section, we will explain our new approach LHiREPS. We begin with an overview of the optimization process. Like HiREPS, we want to optimize a 'mixed options' policy. Although, unlike HiREPS, we want to reliably find multiple solutions. To enforce that multiple options should be learned we will add a new constraint to the optimization problem. However, rather than adding this constraint to the already quite large HiREPS optimization problem, we choose to split up the learning process into a two level hierarchy. For the gating policy, we solve an optimization problem and for each individual option policy we solve an optimization problem. This hierarchy nicely mimics the hierarchical nature of our policy and -as we will see later- makes it easy to add our new constraint to the gating optimization.

An overview of the learning process can be seen in Figure 4.1.

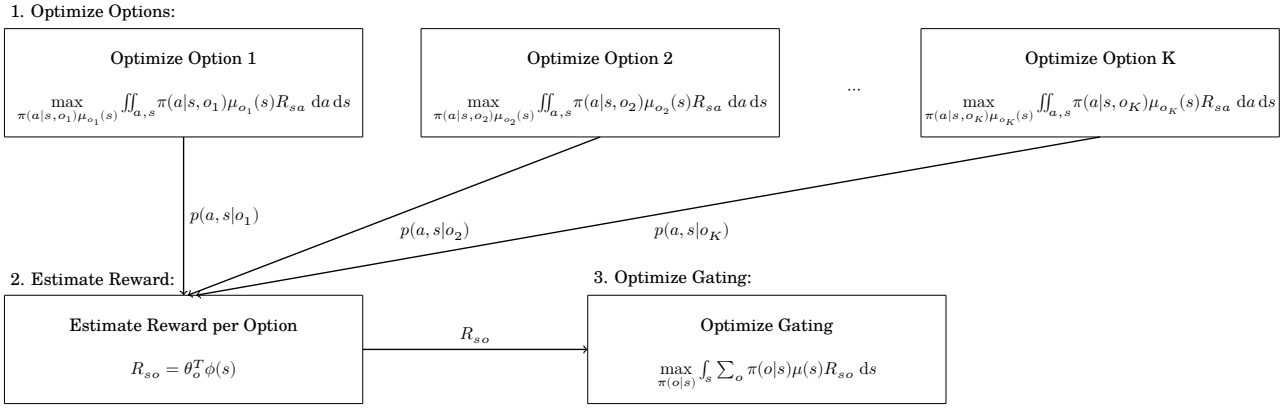


**Figure 4.1.:** An overview of how the gating and option policies are updated. The option policies $\pi(a|s,o)$ are updated first. The reward $R_{so}$ is estimated second and the gating policy $\pi(o|s)$ is updated last. The constraints for the option and the gating optimization problems are omitted. $K$ is the total number of options.

We can see that the first step is to optimize all option policies. This optimization will give us the optimal probability $p(a,s|o)$ which we use to update the parametric model of the option policy, but we also use them to estimate the reward $R_{so}$. This reward is necessary because, to optimize the gating policy, we want a reward with respect to state-option pairs rather than state-action pairs. With $R_{so}$ estimated we can optimize the option policies. All these optimization procedures are done on the whole set of samples, which is especially important for the option policy optimizations. The option policies are optimized with all generated samples and not just with the samples that were generated by the specific option being optimized. In a certain sense this sample sharing allows the options to learn from the successes and failures of each other.
Furthermore, it can make sense to repeat this whole process. The reason is that changing the gating probability $p(o|s)$ will change the result of the option policy optimization $p(a,s|o)$. These new results can be used again to estimate $R_{so}$ and $p(o|s)$, so this optimization process can be repeated many times. However, this repetition increases the required time, which will generally only be useful if the sampling process takes even longer. If that is not the case, then using more iterations may yield better result than repeating the optimizations.

In the following three sections, we will explain the three steps from figure 4.1 in more detail.

## 4.2 Option Policy Optimization

First, we take a look at how to optimize the option policies $\pi(a|s,o)$. Even though we optimize only part of the whole policy (only one option) we can still use the typical REPS structure, discussed in the Foundation chapter. We define the problem as a constraint optimization problem with the typical REPS constraints:

- We only want to make small steps in each iteration so we bound the KL-divergence.
- We want our estimated joint probability $p(a,s)$ to respect the state distribution, so the expectation of the state features has to match the observed average state features.
- We want the joint to be a proper probability distribution, so we require it to sum to 1.

Beyond these constraints, we also want -like HiREPS- that our options are diverse because obtaining the same option policy twice is useless. Here, we have to first consider in which optimization step we want to enforce the diversity constraint. Is it better to enforce the diversity in the gating optimization or in the option optimization? It would be nicer to add this constraint to the gating optimization step because diversity is not a property of a single option but a property of a group of options. The gating optimization would be better suited for this property as it optimizes over all options. However, the gating optimization only optimizes which option policy to choose and has no control over the actions of these option policies. Without this control, we cannot add the diversity constraint to the gating optimization. We have to add it to the option policy optimization problem.

To enforce diversity in the option policies, we use the same constraint as HiREPS, which is given by

$$\kappa \geq -\iint_{a,s} p(a,s) \sum_{o} p(o|s,a) \log p(o|s,a) \, \mathrm{d}a \, \mathrm{d}s.$$

As explained in the HiREPS section, this constraint forces the option policies to specialize. However, it does not allow for a closed form solution for the policy of HiREPS and we have the same problem.
One solution to the problem is to do what HiREPS does, i.e., estimating $p(o|s,a)$ at the start of an iteration and not changing it during the optimization. This process works for HiREPS because it is an EM-style algorithm that optimizes a lower bound. For the option policies, we can also follow this process because we also get a lower bound from keeping $p(o|s,a)$ fixed (see appendix A.4 for proof).
That said, we choose a different approach. Rather than keeping $p(o|s,a)$ fixed throughout the optimization, we say $p(o|s,a)$ is a new variable which is independent of $\pi(a|s,o)$. The fact that $p(o|s,a)$ is now independent of $\pi(a|s,o)$ lets us find a closed form solution for $p(a,s|o)$. There is one obvious problem, though: The probability $p(o|s,a)$ actually depends on $\pi(a|s,o)$. We solve this problem by adding this dependence as a constraint to the optimization problem. The dependency we want to have is

$$\hat{p}(o|s,a) = \frac{p(a,s,o)}{p(a,s)},$$

where $\hat{p}(o|s,a)$ is our new variable which only depends on $\pi(a|s,o)$ through the new constraint we are introducing.
We add this new constraint in the form

$$p(a,s,o) = \hat{p}(o|s,a)p(a,s).$$

This equation is equivalent to the previous equation, but allows for easier derivation of the Lagrangian. Of course, this constraint is not one constraint, but actually an infinite number of constraints. However, when we later approximate the dual function we effectively get one constraint for each sample and option.
Using this constraint should be more effective at enforcing diversity compared to the EM-style approach as $p(o|s,a)$ will actually change according to how $\pi(a|s,o_j)$ changes during the optimization.

The optimization problem for the $j$-th option $\pi(a|s,o_j)$ is now given by

$$\begin{aligned}
\max_{\pi(a|s,o_j)\mu_{o_j}(s),\hat{p}(o|s,a)} & \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)R_{sa}\,\mathrm{d}a\,\mathrm{d}s, \\
\text{s.t:}\quad \varepsilon_{o_j} &\geq D_{\mathrm{KL}}\Big(\pi(a|s,o_j)\mu_{o_j}(s)||q(a,s|o_j)\Big), \\
\kappa &\geq -\iint_{a,s} p(a,s)\sum_o p(o|s,a)\log\hat{p}(o|s,a)\,\mathrm{d}a, \\
p(a,s,o) &= \hat{p}(o|s,a)p(a,s), \\
\hat{\phi}_{o_j} &= \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\phi(s)\,\mathrm{d}a\,\mathrm{d}s, \\
1 &= \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\,\mathrm{d}a\,\mathrm{d}s.
\end{aligned}$$

The distribution $\mu_{o_j}(s) = p(s|o_j)$ denotes the state distribution given the $j$-th option and $\hat{\phi}_{o_j}$ is the observed average feature for the $j$-th option.

Solving the optimization problem for $p(a,s|o_j)$ with the method of Lagrangian multipliers yields the solution

$$p(a,s|o_j) \propto q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+q(o_j)\Big(\xi_{o_j}\log\hat{p}(o_j|s,a)+\sum_o[\zeta_{o_j}(a,s,o)\hat{p}(o|s,a)]-\zeta_{o_j}(a,s,o_j)\Big)}{\eta_{o_j}}\right).$$

The parameters $\eta_{o_j}$, $\theta_{o_j}$, $\xi_{o_j}$, $\zeta_{o_j}(a,s,o)$ are obtained by optimizing the dual function. Like REPS and HiREPS the parameter $\eta_{o_j}$ scales our offset reward and the term $\theta_{o_j}^T\phi(s)$ offsets the reward depending on the state. We also subtract the self-information of the responsibilities $-\xi_{o_j}\log\hat{p}(o_j|s,a)$ similar to HiREPS, but unlike HiREPS we also scale it with $q(o_j)$. This scaling is because we optimize for $p(a,s|o)$ rather than $p(a,s,o)$. Completely new is the $\zeta_{o_j}$-part we get from our new constraints, which is not as easily interpretable as the other terms that offset the reward because $\zeta_{o_j}(a,s,o)$ can change the exponent arbitrarily. However, we can at least see that this freedom is important because in a given situation the constraint $p(a,s,o)=\hat{p}(o|s,a)p(a,s)$ may only be violated for some samples but not all, making it necessary that the Lagrangian multiplier $\zeta_{o_j}$ adapts to these samples.
We find these parameters by minimizing the dual function (for the derivation see appendix A.2) for $\eta_{o_j}>0$ and $\xi_{o_j}>0$. The dual function is given by

$$\begin{aligned}
&g(\eta_{o_j},\theta_{o_j},\xi_{o_j},\zeta_{o_j}) = \\
&\eta_{o_j}\log\left(\iint_{a,s} q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+q(o_j)\Big(\xi_{o_j}\log(\hat{p}(o_j|s,a))+\sum_o[\zeta_{o_j}(a,s,o)\hat{p}(o|s,a)]-\zeta_{o_j}(o_j,a,s)\Big)}{\eta_{o_j}}\right)\mathrm{d}a\,\mathrm{d}s\right) \\
&+\eta_{o_j}\varepsilon_{o_j}+\xi_{o_j}\kappa+\theta_{o_j}^T\hat{\phi}_{o_j}+\sum_{o_k\neq o_j}\iint_{a,s} q(a,s|o_k)q(o_k)\left(\xi_{o_j}\log(\hat{p}(o_k|s,a))+\sum_o[\zeta_{o_j}(a,s,o)\hat{p}(o|s,a)]-\zeta_{o_j}(a,s,o_k)\right)\mathrm{d}a\,\mathrm{d}s.
\end{aligned}$$

We can see the dual function still contains $\hat{p}(o|s,a)$ as we cannot compute a closed form solution of $\hat{p}(o|s,a)$. We cannot compute one because our extremum for $\hat{p}(o|s,a)$ is

$$\hat{p}(o|s,a) = -\frac{\xi_{o_j}p(o|s,a)}{\zeta_{o_j}(a,s,o)}, \qquad \text{where} \quad p(o|s,a) = \frac{p(a,s|o)q(o)}{\sum_{o_k}p(a,s|o_k)q(o_k)}.$$

For $o=o_j$ we have $p(a,s|o_j)$ in the denominator, which is our closed form solution for $p(a,s|o_j)$. We can see that it depends again on $\hat{p}(o_j|s,a)$ in complicated ways.

Without a closed form solution, we cannot compute $\hat{p}(o|s,a)$ easily. Instead, we use the derivative of the Lagrangian with respect to $\zeta_{o_j}$ and solve it for $\tilde{p}(o|s,a)$, which gives us the equation

$$\hat{p}(o|s,a) = \frac{p(a,s,o)}{p(a,s)}.$$

We use this equation to update $\hat{p}(o|s,a)$ during the optimization.

To update the value of $\zeta_{o_j}$ we do the opposite. We take the derivative of the Lagrangian with respect to $\tilde{p}(o|s,a)$ (see appendix A.3 for derivation) to get

$$\zeta^*_{o_j}(a,s,o) = -\frac{\xi_{o_j} p(o|s,a)}{\hat{p}(o|s,a)}.$$

We use the value of $\zeta^*_{o_j}$ to compute a gradient

$$g_{\zeta_{o_j}}(a,s,o) = (\zeta_{o_j}(a,s,o) - \zeta^*_{o_j}(a,s,o)),$$

and then use line search with $\zeta_{o_j}(a,s,o)$ as starting point and $g_{\zeta_{o_j}}(a,s,o)$ as search direction to find the value for $\zeta_{o_j}$ that minimizes the dual function along our search direction. The minimum is our updated $\zeta_{o_j}$ value.
The experiments show that this update rule works half well. On the low dimensional task in our experiments section, it leaves little to be desired in terms of diversity, but in terms of convergence it is a lot slower than HiREPS. On the higher dimensional task from our experiment section, it converges not too much slower than HiREPS and is on average more diverse.
We also explored the more intuitive possibility of using the derivative of the Lagrangian w.r.t. $\hat{p}(o|s,a)$ to update $\hat{p}(o|s,a)$ and the derivative of the Lagrangian w.r.t. $\zeta_{o_j}$ to update $\zeta_{o_j}$, but the optimization took much longer and did not yield better results.

There is one more thing which we have to watch out for: The expected entropy constraint and the KL-divergence constraint may be impossible to fulfill at the same time because we would have to change the policy too much to decrease the expected entropy below $\kappa$. This issue can be solved by increasing $\kappa$ whenever the diversity constraint is not satisfied after the optimization. With the increased $\kappa$ we retry optimizing the dual.

To update our policies, we also minimize $D_{\mathrm{KL}}\left(p(a,s|o_j)||\pi_m(a|s,o_j)\mu_{o_j}(s)\right)$ for $\pi_m(a|s,o_j)$ similar to REPS. This minimization also turns into a weighted maximum likelihood estimation.

### 4.3 Option Reward Estimation

Next, we want to optimize our gating policy $\pi(o|s)$ so that it maximizes the expected reward of state-option pairs $E[R_{so}]$. To do this optimization, we have to first estimate the reward $R_{so}$ itself. The model we use for $R_{so}$ is

$$R_{so} = \theta_o^T \phi(s),$$

where $\theta_o$ is the parameter we will estimate and $\phi(s)$ is the feature vector we will use in the gating policy. This is important. As we will see in the next section, this structure will enable us to update the gating policy directly. Additionally, it will also allow us to remove one constraint from the constrain optimization problem.

To estimate $R_{so}$ we optimize $\theta_o$ with weighted maximum likelihood, i.e,

$$\theta_o^* = \mathrm{argmin}_{\theta_o} \sum_{i=1}^{N} p\left(a^{[i]}, s^{[i]}|o\right)\left(R_{sa}^{[i]} - \theta_o^T \phi\left(s^{[i]}\right)\right)^2.$$

Here, the probabilities $p(a^{[i]}, s^{[i]}|o)$ are the ones we estimated in the previous step when we optimized the option policies. The index $(\cdot)^{[i]}$ denotes the i-th sample and $N$ is the total number of samples.

With the reward $R_{so}$ we can now update the gating policy.

### 4.4 Gating Policy Optimization

In this section, we will look at how to optimize the gating. We want to optimize it in such a way that we do not have the problem of HiREPS, which was that we generally end up with only few or no alternative solutions. However, before we look at how to make the options keep their probability, we first look at how we can maximize the expected reward.
We already know how a policy can be optimized with REPS. So the REPS optimization problem will be the starting point. The REPS optimization problem for $p(o,s)$ is given by

$$\begin{aligned}
&\max_{\pi(o|s)\mu(s)} \int_s \sum_o \pi(o|s)\mu(s)R_{so}\ \mathrm{d}s,\\
&\text{s.t:}\ \ \varepsilon \geq D_{\mathrm{KL}}\left(\pi(o|s)\mu(s)||q(o,s)\right),\\
&\qquad \hat{\phi} = \int_s \sum_o \pi(o|s)\mu(s)\phi(s)\ \mathrm{d}s,\\
&\qquad 1 = \int_s \sum_o \pi(o|s)\mu(s)\ \mathrm{d}s.
\end{aligned}$$

Using this optimization problem we can optimize the expected reward as explained in the Foundation chapter. However, we can make the optimization simpler by leaving out the $\phi$-constraint. This constraint can be left out because we will not optimize for the joint $p(o,s)$ but optimize for the policy $\pi(o|s)$ directly. The reason we can use $\pi(o|s)$ is because of the special form of our reward $R_{so}$ and the structure of our gating policy. We will look at that later when we discuss how to update the policy after the optimization.

With the basic setup in place, we now take a look at how to make the options keep their probability. The problem we have so far is the same as HiREPS: There is no trade-off or constraint that prohibits one option from having 100% probability and all others having 0% probability. This freedom means whenever one option is better in every state than any other option, then this one option will end up with 100% probability. With only one option being optimized, we end up with no backup options.
We can avoid this problem by forcing the policy $\pi(o|s)$ to keep the options more uniformly distributed. We can restrict the probabilities to be more uniform by setting a lower bound on the expected entropy of the policy $\mathrm{E}_s[\mathrm{H}_\pi[o|s]]$ because -as discussed in the HiREPS section- the higher the Entropy is the more uniform the distribution has to be. With this constraint, we can enforce that at least $K$ options retain some probability. We achieve this goal by setting the lower bound of the constraint to a value slightly above the logarithm of $K-1$. The reason this setting works is that the highest value of $\mathrm{E}_s[\mathrm{H}_\pi[o|s]]$ for $K-1$ options is $\log(K-1)$. This value is only achieved when all $K-1$ options are uniformly distributed. When we set the value of the lower bound slightly higher than $\log(K-1)$ the only way the constraint can be satisfied is if more than $K-1$ options have some probability. This insight gives us a nice way to control the number of backup options we want to have.

Now, we can formulate the constraint optimization problem for the gating policy. It is given by

$$\begin{aligned}
&\max_{\pi(o|s)} \int_s \sum_o \pi(o|s)\mu(s)R_{so}\ \mathrm{d}s,\\
&\text{s.t:}\ \ \varepsilon \geq \mathrm{E}_s[D_{\mathrm{KL}}\left(\pi(o|s)||q(o|s)\right),\\
&\qquad \alpha \leq \mathrm{E}_s[\mathrm{H}_\pi[o|s]],\\
&\qquad 1 = \int_s \sum_o \pi(o|s)\mu(s)\ \mathrm{d}s.
\end{aligned}$$

Solving the optimization problem with the method of Lagrangian multipliers yields a closed form solution $\pi(o|s)$

$$\pi(o|s) \propto q(o|s)^{\frac{\eta}{\eta+\beta}} \exp\left(\frac{R_{so}}{\eta+\beta}\right).$$

The parameters $\eta$ and $\beta$ are Lagrangian multipliers. We can see that $\eta$ controls how close the new policy $\pi(o|s)$ is to the old policy $q(o|s)$. The higher $\eta$ is, the closer $\pi(o|s)$ is to $q(o|s)$.

On the other side we have $\beta$, which is the Lagrangian multiplier for the new constraint $\alpha \leq \mathrm{E}_s[\mathrm{H}_\pi[o|s]]$ and controls how uniform the distribution $\pi(o|s)$ will be. The higher $\beta$ is, the more uniform $\pi(o|s)$ becomes.
We get $\eta$ and $\beta$ by optimizing the dual function

$$g(\eta,\beta) = (\eta+\beta)\log\left(\int_s \sum_o (q(o|s))^{\frac{\eta}{\eta+\beta}}\, \mu(s) \exp\left(\frac{R_{so}}{\eta+\beta}\right) \mathrm{ds}\right) + \eta\epsilon - \alpha\beta$$

for $\eta > 0$ and $\beta > 0$. The derivation can be found in the appendix A.5.

To update our policy, we do not minimize the KL-divergence this time. Instead -as mentioned earlier- we can compute and update $\pi(o|s)$ directly. To understand how we can update $\pi(o|s)$ directly, we just have to remind ourselves of the structure of $R_{so}$ and $q(o|s)$ and insert them in the closed form solution:

$R_{so}$ models the reward per option which we estimated in the previous section. It is given by

$$R_{so} = \theta_o^T \phi(s).$$

The distribution $q(o|s)$ is a softmax which we can simplify by using

$$q(o|s) = \frac{\exp(\bar{\theta}_o^T \phi(s))}{\sum_o \exp(\bar{\theta}_o^T \phi(s))} \propto \exp(\bar{\theta}_o^T \phi(s)).$$

Inserting them both in the closed form solution for the new gating policy gives us

$$\pi(o|s) \propto \exp\left(\left(\frac{\eta\bar{\theta}_o + \theta_o}{\eta+\beta}\right)^T \phi(s)\right).$$

We see that the solution is proportional to a softmax of the same form $\exp(\theta^T\phi(s))$. This form allows us to update $\pi$ by updating $\bar{\theta}_o$ to its new value $\bar{\theta}_o^*$ with the equation

$$\bar{\theta}_o^* = \frac{\eta\bar{\theta}_o + \theta_o}{\eta+\beta}.$$

This direct update spares us from calculating a weighting, which in turn allows us to skip the $\phi$-constraint. However, skipping the $\phi$-constraint is just a nice bonus. It is also possible to calculate weightings $p(o,s)$ in the normal REPS style by including the $\phi$-constraint.

### 4.5 Relation to REPS

If we use a single option $o_1$, then using LHiREPS becomes equivalent to using the REPS algorithm. This equality is because the option policy $\pi(o|s)$ is always 1 for the only option we have, meaning the gating optimization and $R_{so}$ estimation are irrelevant. For the option policy the probability $\hat{p}(o_1|s,a)$ and $q(o_1)$ are always 1 and $q(a,s|o_1)$ is equal to $q(a,s)$ for one option. With these values the closed form solution simplifies to that of REPS.
The dual function becomes nearly identical to that of REPS with the exception of the term $\xi_{o_1}\kappa$ being added to it. However, the other Lagrangian multipliers and $p(a,s,o)$ are independent of the value of $\kappa$ or $\xi_{o_1}$ and so the solution for $\eta_{o_1}$ and $\theta_{o_1}$ are the same as optimizing the REPS dual function.

# 5 Experiments

## 5.1 Illustration

We first examine some of the properties of LHiREPS on a simple toy task and then look at a more complex task. The toy task can be seen in Figure 5.1. Since the reward function is a mixture of Gaussians we will refer to this task as the Gaussian Task. We can see the Gaussian Task is a one dimensional task with a few modes. We will remove some of these modes to see if LHiREPS actually learned backup solutions and compare the backup solutions against those that HiREPS learned, but first we will explain the setup.

For all experiments we carried out 20 trials. Each trial ran 70 iterations and in each iteration we generated 500 new samples and used them with the 500 samples generated in the previous iteration to train our algorithm. The plots show the average over these trials as lines and the standard deviation of the trials as shaded regions around the lines. The policy we trained was a 'mixed options' policy, which we described in the Foundation chapter. Examples of the initialization of our option policies can be seen in Figure 5.1 and Figure 5.3. The gating policy was initialized to be uniformly distributed. For LHiREPS and HiREPS we used the following parameters: For HiREPS we used $\varepsilon = 0.5$ and $\kappa$ was set to the observed expected Entropy of $q(o|s,a)$ i.e. $\mathrm{E}_{s,a}[\mathrm{H}_q[o|s,a]]$. For our algorithm in the gating optimization we set $\varepsilon = 0.5$ and $\alpha = \log(K-0.5)$, where $K$ is the number of options. In the option optimization we set $\varepsilon_{o_j} = 0.5$ for all options $j$ and $\kappa$ was set like in HiREPS. We used both variants of our algorithm. The variant with the constraint $p(a,s,o) = \hat{p}(o|s,a)p(a,s)$ as well as the EM variant. We will refer to these just as non-EM and EM variant. The number of options we learned was 10, but we also trained with more and less options and we will look at the differences between the number of options later. Regarding the parameters we also tried different values and the mentioned values were the best. As base for the logarithms we used $e$.

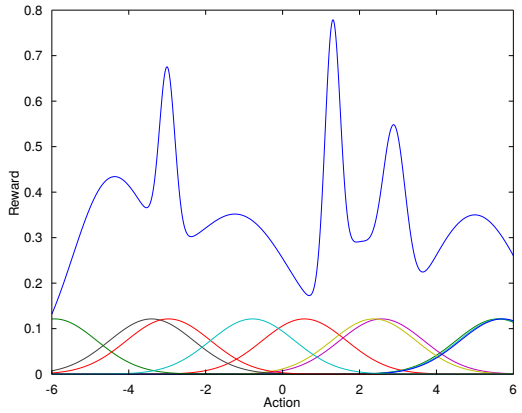


**Figure 5.1.:** The reward function in blue with the different option policies. The plot shows the random initialization of the option policies for 10 options for one of the trials.

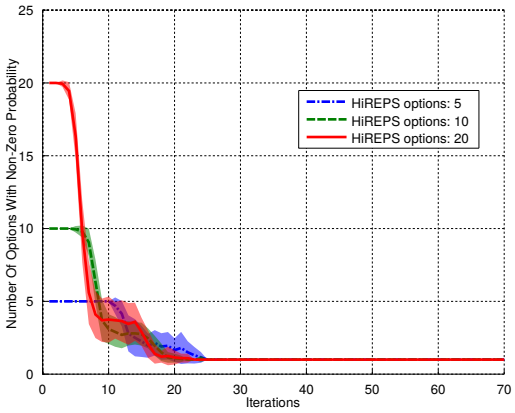


**Figure 5.2.:** The number of options that have non-zero probability for the HiREPS algorithm on the Gaussian Task plotted over the iterations.

Before we look at diversity it is interesting to first examine how good the best solution of our algorithm is and how it compares to that of HiREPS. In Figure 5.4 we have plotted the normalized average reward for each iteration of the best option, where we mean by "best option" the option that had the highest average reward in the last iteration.
What we can see is that all three algorithms find the best reward, but HiREPS and the EM variant of our approach converge much faster to the maximum reward than the non-EM variant. The fact that the EM approaches are better makes sense as they replace the real $p(o|s,a)$ with $\tilde{p}(o|s,a)$ in the constraints, allowing them to be less diverse in favor of being more greedy. Another reason is also that the dual function of HiREPS and our EM variant are a lot easier to optimize numerically than the dual of the non-EM variant. Further HiREPS has the advantage that it has no constraint to keep

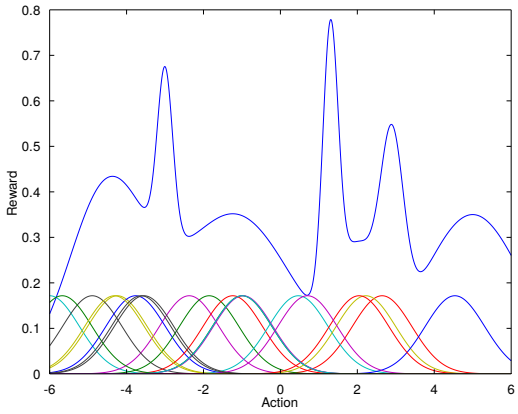


**Figure 5.3.:** The reward function in blue with the different option policies. The plot shows the random initialization of the option policies for 20 options for one of the trials.

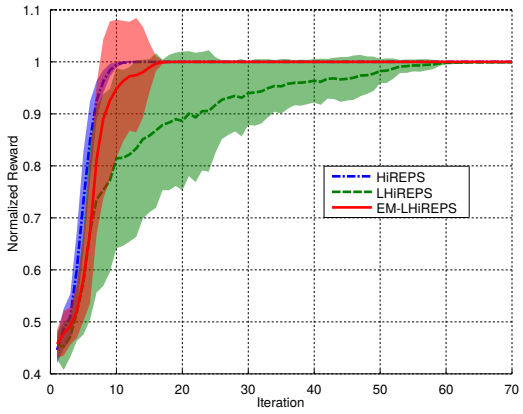


**Figure 5.4.:** The normalized reward for the best option on the Gaussian Task from Figure 5.1 plotted for HiREPS, our EM variant and our non-EM variant.

the probability of options more uniform, meaning the good options will have higher probability and thus more samples. This can be seen in Figure 5.2, where we plotted the number of options with non-zero probability for HiREPS. After less than ten iterations many of the options have zero probability. For our approach we choose a high $\alpha$ value to keep some probability for all options.

Next we look at how well LHiREPS achieves our goal of finding backup solutions. We test the diversity of the found solutions by training HiREPS and our new approach on the Gaussian Task, but evaluating them on an altered version of the task (seen in Figure 5.5). The altered version is missing the highest mode. In Figure 5.6 we again plotted the normalized average reward for the best option on the altered task. We can see that the best option of our new approach in both variants always converged to the maximum, whereas the best option of HiREPS for this case does not converge to it. The reason is that generally all options except one have zero probability (see again Figure 5.2). The only option that does not have zero probability is the one that converged to the maximum of the Gaussian Task which is missing in the altered version. The other options cannot reliably converge to other modes before they end up with zero probability and so the second-best option is not always found.

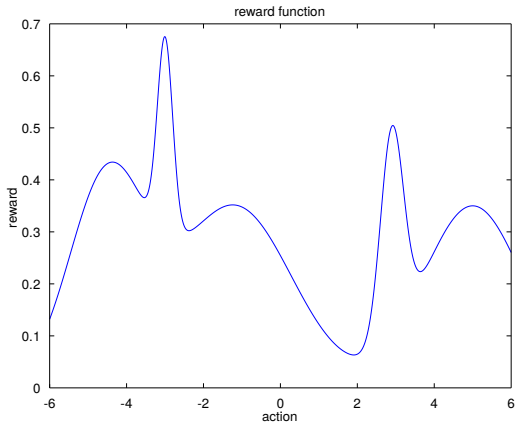


**Figure 5.5.:** The reward function of our simple Gaussian Task without the highest mode.

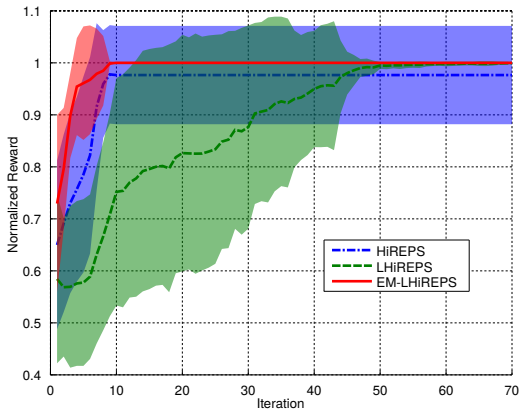


**Figure 5.6.:** The normalized reward for the best option on the modified Gaussian Task (Figure 5.5) after training on the normal Gaussian Task (Figure 5.1) plotted for HiREPS, our EM variant and our non-EM variant.

The same is true if we train and evaluate in the same manner as before, but this time remove the best and second-best mode (see Figure 5.7). In Figure 5.8 we can see the average reward of the best option for both algorithm. Both LHiREPS

variants find the third-best mode often but not always. However, the difference to HiREPS is a lot more clear now, as it never finds the third-best mode. We can also notice here that on average the non-EM variant is slightly more diverse than the EM-variant.

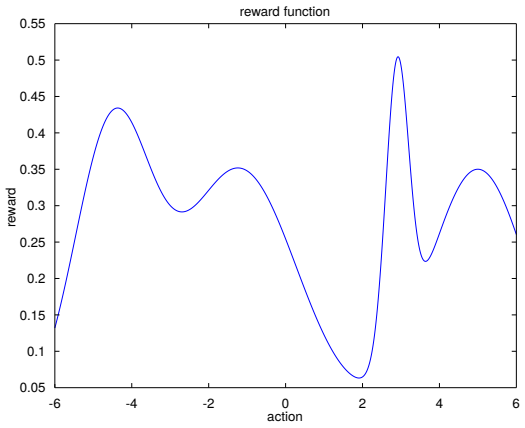


**Figure 5.7.:** The reward function of our simple Gaussian Task without the two highest modes.

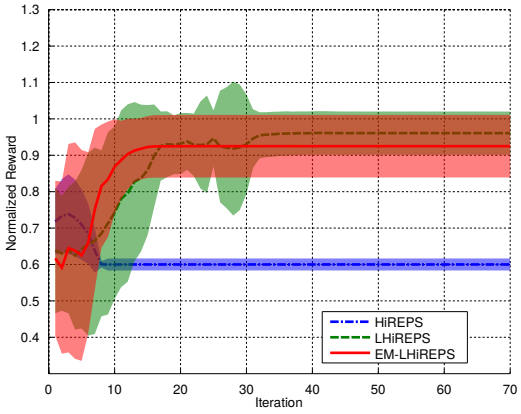


**Figure 5.8.:** The normalized reward for the best option on the modified Gaussian Task (Figure 5.7) after training on the normal Gaussian Task (Figure 5.1) plotted for HiREPS, our EM variant and our non-EM variant.

Lastly we want to quantify how many of the modes of our Gaussian task were found. Instead of continuing to go over each individual mode, we try to measure the overall diversity. We measure diversity by summing over the rewards of the options with the highest reward per mode. In other words, we evaluate all options on only one of the Gaussians of our Gaussian Task and then for each of the Gaussians we select the highest reward achieved for this Gaussian. All these highest rewards are normalized by dividing by the maximum value of the respective Gaussian. We then take the average over the modes. After this procedure we get a useful measure of diversity. When the options are all converged to the same Gaussian, the maximum reward would be about $1/6$ because only 1 out of the 6 Gaussians was found[1], whereas if for each Gaussian there is an option that sits exactly on the mode the overall score would be $6/6$ because all modes were found.

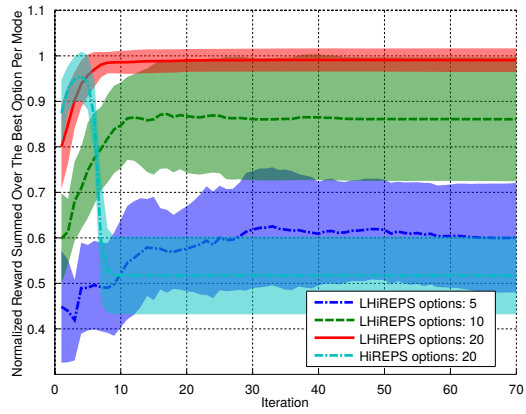


**Figure 5.9.:** The normalized reward summed over the best option per mode plotted for each iteration of our non-EM variant with options 5, 10 and 20 and of HiREPS with 20 options.

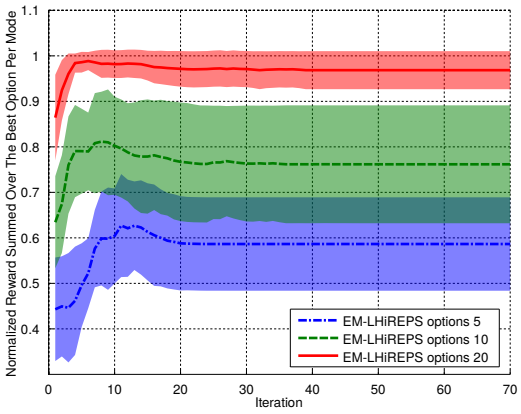


**Figure 5.10.:** The normalized reward summed over the best option per mode plotted for each iteration of our EM variant with options 5, 10 and 20.

[1] Even when all options converge to one Gaussian, the score is a little bit higher than $1/6$. The reason is that the options will also get a some score for being in the tail of another Gaussian.

This diversity score is plotted in Figure 5.9 for the non-EM variant and the HiREPS trials that started with 20 options (which were the most diverse HiREPS trials). We also plotted the diversity for the EM variant of our approach, which can be seen in Figure 5.10.
What we can see here is that the diversity of the EM variant rises fast, but always decreases again between the 10th and 20th iteration. The non-EM variant on the other hand does not decrease again generally and also finds more modes on average. The difference is especially large for 10 options where on average the non-EM variant reaches up to 85% and the EM variant only reaches a bit above 75%. This difference indicates that our odd treatment of $\hat{p}(o|s,a)$ and $\zeta_{o_j}$ in our dual function works with respect to diversity (not so much with respect to fast convergence). In the end, for this task the EM variant seems to be a good compromise between the speed of HiREPS convergence and the diversity achieved by our non-EM variant.

## 5.2 Evaluation

We use a planar reaching task to evaluate our algorithm. 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 reaching points are at time 50 and the second reaching points are at time 100. They can be seen in Figure 5.11 as blue and red dots respectively. At each of these times the robot has to only reach one of the many possible reaching points. Figure 5.11 shows one possible movement of the robot. A score is given based on the squared distance of the end effector to the closest reaching point at the specified time (50 or 100). The reward is the sum of these scores plus a penalty for high acceleration.
The robots initial position has all joints aligned pointing to the left with a random offset of 0.5 radians ($\approx 29°$) up or down(see Figure 5.12). This randomness makes this task state-dependent. The only part of the system state we pass to the algorithm as the environments state $s$ is the value of the angle that is randomly varied. To generate a trajectory we use Dynamic Movement Primitives (DMP)[13]. We have one DMP for each joint. The DMPs generate the angle of the joints. The goal position is set so that the arm points to the right. Each DMPs use five basis function. The linear weights of these five basis functions are the actions our options policies generate. To track the generated trajectory we use a PD controller with a feedforward term [14].

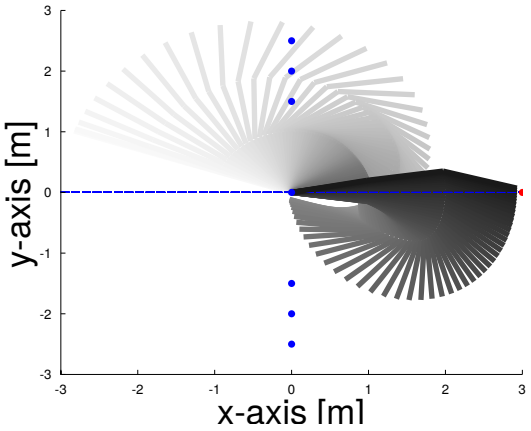


**Figure 5.11.:** A trajectory that reaches one of the reaching points at the correct time steps. The robot arm is drawn for each time step. The shade of gray indicates which time step: The lighter the gray the closer to zero, the darker the gray the closer to the final time step. The blue dots are the points that have to be reached at time 50 and the red dot is the point that has to be reached at time 100.

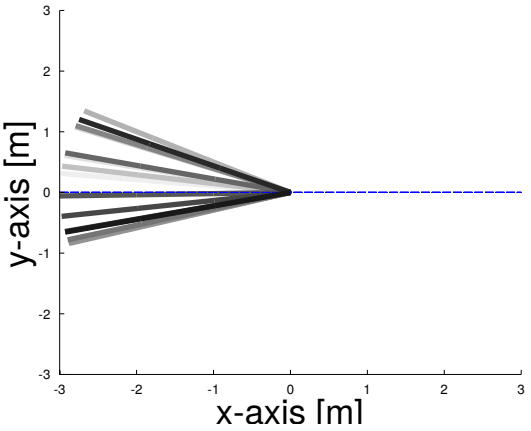


**Figure 5.12.:** 15 different samples of the initial position of the robot arm in 15 shades of gray.

For all experiments 20 trials were used with each trial consisting of 100 iterations. In each iteration 500 new samples were generated and the algorithms were trained on the 500 samples generated in the previous iterations and the 500 newly generated samples. The plots show the average over these trials as lines and the standard deviation of the trials as shaded regions around the lines. The policy we trained was a 'mixed options' policy and the parameters used for the algorithms were the same as the ones for the Gaussian Task explained in the Illustration section.
As features for the options policies we just used the state:

$$\phi_1(s) = s$$

As features for the gating policy and our algorithm we used squared features

$$\phi_2(s) = \begin{pmatrix} 1 \\ s \\ s^2 \end{pmatrix}$$

The number of options we used were 2, 5 and 10.

Like in the previous section, we will first examine the best option performance of each algorithm. In Figure 5.13 we have 4 plots showing the reward for the best option. The first three compare the three algorithms for different numbers of options. The fourth plot compares different numbers of options for just the non-EM variant. Looking at these plots we can see that the difference between our EM and our non-EM variant is very small for this task. For two options the EM-variant is slightly better, but for five and ten options the non-EM variant is slightly better.

We can also see that the difference between HiREPS and our two variants is not that big for two options, but becomes progressively bigger the more options we use. For 10 options HiREPS is slightly better than our two variants. Further it also seems that using more options improves the learning speed of our algorithm.

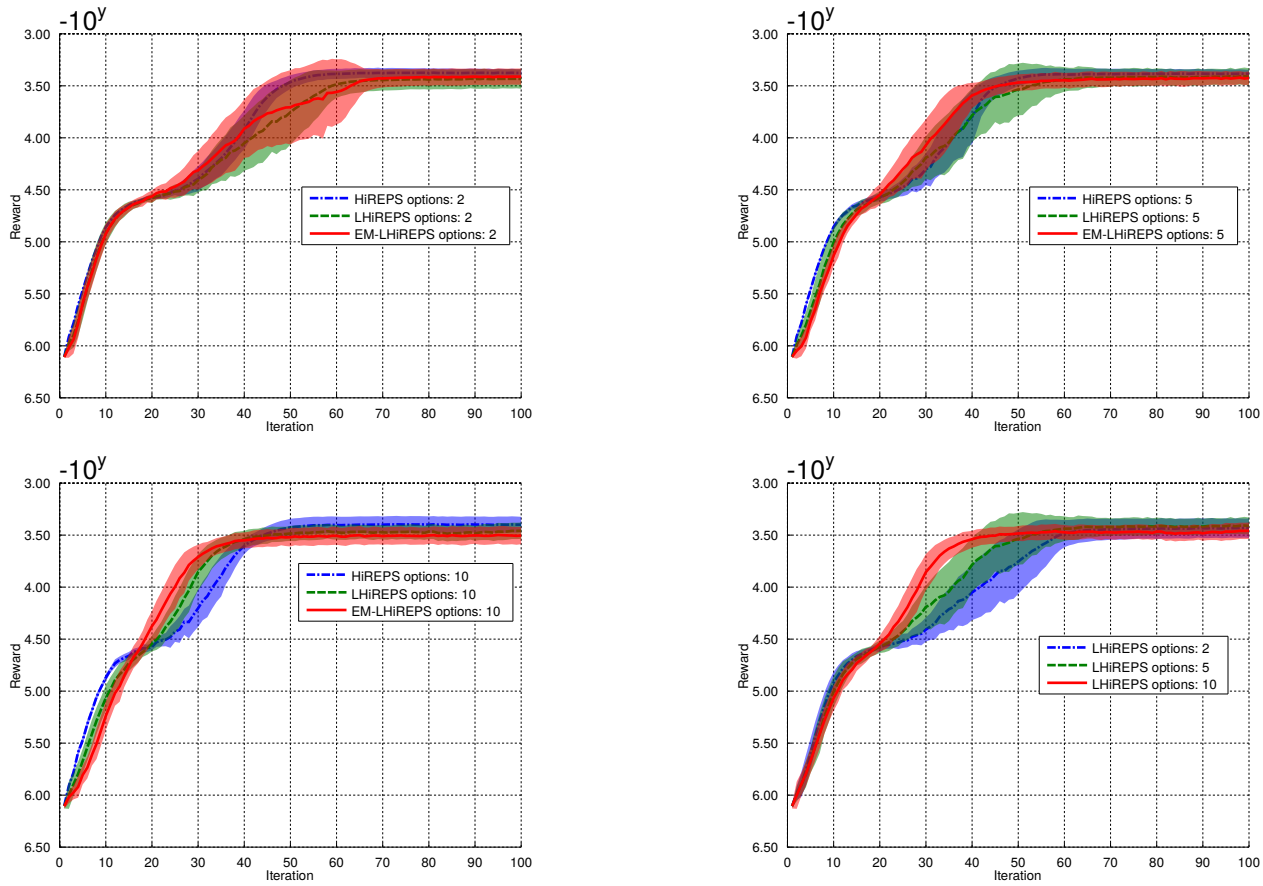


**Figure 5.13.:** The plot in the lower right shows the best option for just the non-EM variant for different numbers of options. The other three plots show the best option of each algorithm for a certain number of options

Now we want to look at the diversity of the solutions found. To asses diversity we removed one of the reaching points at time 50 and evaluated all the options we found after training on the remaining reaching points. For each of these evaluations we took the best option and summed these. Figure 5.14 shows these sums averaged over the trails. The bounds indicate the standard deviation. The cyan, green and yellow plots are for 2, 5 and 10 options respectively. For the evaluation we used 20 samples per option and we removed the acceleration penalty. For HiREPS we only used options that had non-zero probability.

What Figure 5.14 shows is that HiREPS does not benefit from using more options for this task. The reason is again that most options end with zero probability. We can see this issue in Figure 5.15. On the other hand our EM and non-EM improve in diversity the more options are used. For 5 and 10 options our variants are better than HiREPS in terms of average diversity. Comparing our two variants we see that the non-EM variant has much higher standard deviation than the EM variant. The high standard deviation shows that it finds diverse solutions less reliably than the EM variant.

Lastly we want to point out that despite using 1000 samples per Iteration for 100 Iterations not all options converged. This fact may sound odd at first because this setup should be enough for each option. However, each option only uses all 1000 samples in the beginning, when the option distributions still have a lot of overlap. In later iterations when the options diversify the number of effective samples is a smaller. To be precise each option learns more from a sample when $q(o|s,a)$ is large. For very small $q(o|s,a)$ the options learns very little and for $q(o|s,a) = 0$ the sample is effectively not used (this effect can be seen when looking at the dual approximation in appendix A.2).

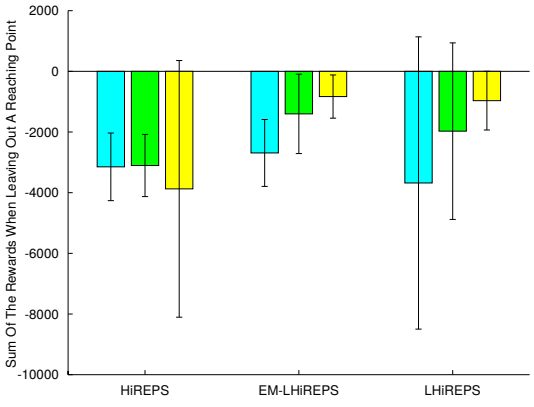


**Figure 5.14.:** This plot is somewhat similar in spirit to the diversity plot in the Illustration section. After training the algorithms for different options, we removed one reaching point at time 50 and evaluated the performance of all options (without the acceleration penalty). This process is done for each of the time 50 reaching points. The reward of the best option for each left out point is summed. The plot shows these sums averaged over the trials. The black bounds show the standard deviation. The different colored plots are from left to right for 2, 5 and 10 options for each algorithm.

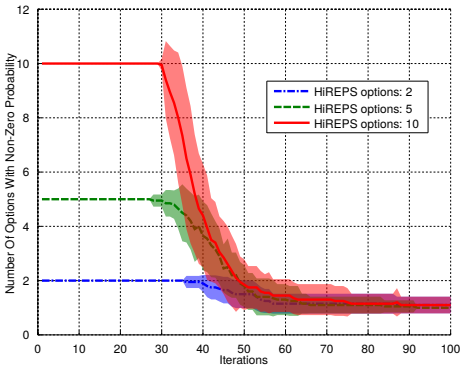


**Figure 5.15.:** The number of options that have non-zero probability for the HiREPS algorithm on the Reaching Task plotted over the iterations.

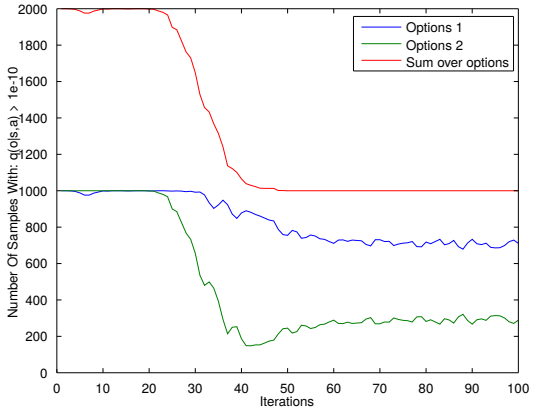


**Figure 5.16.:** The sum of the number of the responsibilities with $q(o|s,a) > 1e-10$ for each option $o$ plotted in green and blue. The sum of the two plots is plotted in red.

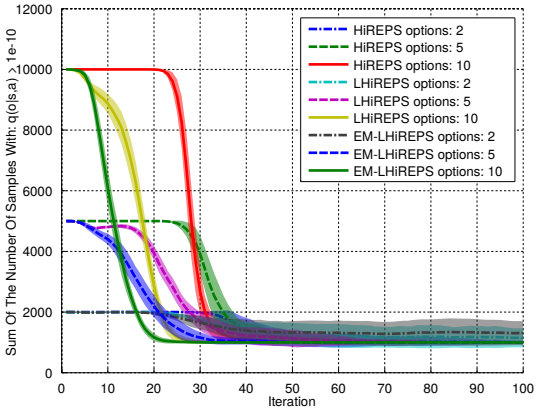


**Figure 5.17.:** The sum over all options of the number of the responsibilities with $q(o|s,a) > 1e-10$ plotted for each algorithm and number of options.

In Figure 5.16 we have plotted for each option $o$ the number of samples with $q(o|s,a) > 1e-10$. In the beginning both options can use effectively 1000 samples, but after 50 iterations option 1 has only about 700 effective samples (samples with $q(o|s,a) > 1e-10$) and option 2 has only about 300. We see the probability $q(o|s,a)$ becomes mutually exclusive. The red graph, which is the sum of the two option graphs, shows this problem best. In Figure 5.17 we have plotted these sums for all algorithms and number of starting options. We can see here that for all algorithms the sum is below 2000 at the 40th iteration. This value means that, for instance, when we have 10 options many of these options will effectively get less than $\frac{2000}{10} = 200$ samples. Of course, the best option will get much more effective samples and the worst options much less. This conflict shows us that there is a trade-off between having more options (with non-zero probability) and finding better options.

# 6 Conclusion and Future Work

In this thesis, we have presented a new algorithm to find versatile solutions. The algorithm itself is a two level hierarchy. The lower level of this hierarchy is responsible for learning the option policies of a 'mixed options' policy and the upper level is responsible for learning the gating policy. Both the upper level optimization as well as the lower level optimizations follow the REPS framework. The lower level optimizations are also very closely related to HiREPS, building on many of HiREPS ideas.

We used this algorithm on a very simple task with 6 local optima. On this task our options diversified enough to find many of the local optima. When we removed the two best local optima, we still had decent backup solutions whereas HiREPS did not. Our algorithm could also find versatile solutions on a more complex planar reaching task. The best solution it found were comparable in terms of reward to those of HiREPS, but it found more diverse solutions than HiREPS. However, of the two variations of our algorithm the non-EM variation was not as reliable as the EM variant in that respect. We also highlighted that there is trade-off between how many options are used and how many effective samples each option receives over the later iterations. In the end, we have seen that having backup solutions certainly helps when the previous optimum disappears. The algorithm we present managed to find such backup solution and the best solution it found are still very good.

In the future, we hope to try this algorithm on more complex tasks. In particular, we want to test if we can learn forehand and backhand strokes in a table tennis simulation.

Regarding more the structure of the algorithm, we think that the partition of the learning algorithm in upper and lower level allows for many extension. Using and adapting this structure to learn hierarchies which are more complex than the 'mixed option' policy, would be interesting.

# Bibliography


[1] C. Daniel, G. Neumann, and J. R. Peters, "Hierarchical relative entropy policy search," in *International Conference on Artificial Intelligence and Statistics*, pp. 273–281, 2012.

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

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[4] 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.

[5] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, 2014.

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

[7] G. Neumann, W. Maass, and J. Peters, "Learning complex motions by sequencing simpler motion templates," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 753–760, ACM, 2009.

[8] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[9] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," *Advances in neural information processing systems*, pp. 1043–1049, 1998.

[10] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *J. Artif. Intell. Res.(JAIR)*, vol. 13, pp. 227–303, 2000.

[11] M. Stolle and D. Precup, "Learning options in reinforcement learning," in *SARA*, pp. 212–223, Springer, 2002.

[12] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, pp. 1281–1288, 2004.

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

[14] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2009.

# A Appendix

## A.1 Trajectories from the Reaching Task

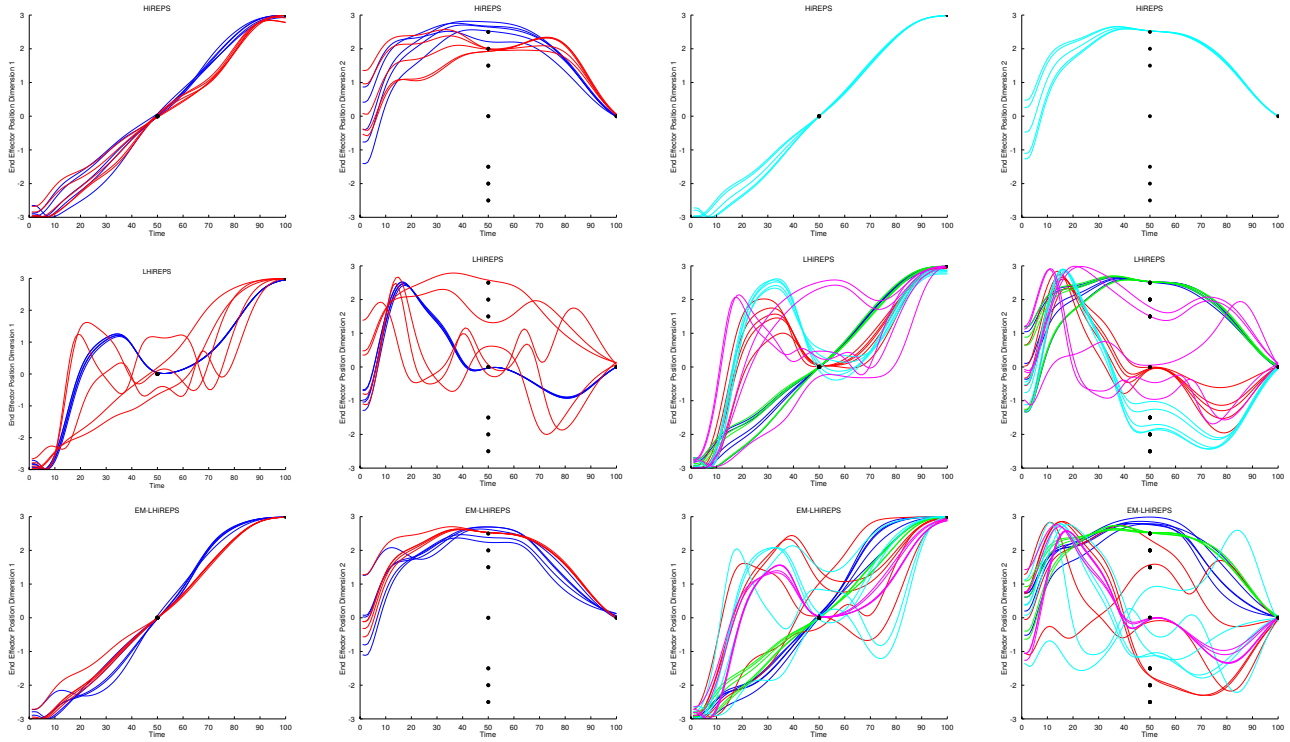


**Figure A.1.:** There are always two plots showing the trajectory of the end effector along the x-dimension and y-dimension respectively. The top row of plots show trajectories generated after learning with HiREPS. The middle row shows trajectories from our non-EM and the bottom row from our EM variant. On the left are always the two plots for a run were the algorithms started with 2 options and on the right are the two plots for a run started with 5 options. The different colors indicate trajectory generated by different options. For each option 5 trajectories are plotted. In the case of HiREPS only options were plotted that had non-zero probability after the learning. The black dots are the reaching points.
We can see that not all options have converged. Also, sometimes options converge to the same reaching point. This convergence is possible if the variance of the Gaussian distribution of one or both of the options is small enough.

## A.2 Option Optimization

The constraint optimization problem for an option is given by

$$\begin{aligned}
\max_{\pi(a|s,o_j)\mu_{o_j}(s),\hat{p}(o|s,a)} & \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)R_{sa}\,\mathrm{d}a\,\mathrm{d}s, \\
\text{s.t:}\quad \varepsilon_{o_j} &\geq D_{\mathrm{KL}}\Big(\pi(a|s,o_j)\mu_{o_j}(s)||q(a,s|o_j)\Big), \\
\kappa &\geq -\iint_{a,s} p(a,s)\sum_o p(o|s,a)\log\hat{p}(o|s,a)\,\mathrm{d}a, \\
p(a,s,o) &= \hat{p}(o|s,a)p(a,s), \\
\hat{\phi}_{o_j} &= \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\phi(s)\,\mathrm{d}a\,\mathrm{d}s, \\
1 &= \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\,\mathrm{d}a\,\mathrm{d}s.
\end{aligned}$$

The distribution $\mu_{o_j}(s) = p(s|o_j)$ denotes the state distribution given the $j$-th option and $\hat{\phi}_{o_j}$ is the observed average feature for the $j$-th option.

The Lagrangian of this optimization problem is

$$\begin{aligned}
L(\pi\mu_{o_j},\eta_{o_j},\xi_{o_j},\zeta_{o_j},\theta_{o_j},\lambda_{o_j}) = &\iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)R_{sa}\,\mathrm{d}a\,\mathrm{d}s \\
&+ \eta_{o_j}\left(\varepsilon_{o_j} - \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\log\left(\frac{\pi(a|s,o_j)\mu_{o_j}(s)}{q(a,s|o_j)}\right)\mathrm{d}a\,\mathrm{d}s\right) \\
&+ \xi_{o_j}\left(\kappa + \iint_{a,s} p(s,a)\sum_o p(o|s,a)\log\hat{p}(o|s,a)\,\mathrm{d}a\,\mathrm{d}s\right) \\
&+ \iint_{a,s}\sum_o \zeta_{o_j}(a,s,o)\left(\sum_{o_k}[p(a,s,o_k)]\,\hat{p}(o|s,a) - p(a,s,o)\,\mathrm{d}a\,\mathrm{d}s\right) \\
&+ \theta_{o_j}^T\left(\hat{\phi}_{o_j} - \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\phi(s)\,\mathrm{d}a\,\mathrm{d}s\right) \\
&+ \lambda_{o_j}\left(1 - \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\,\mathrm{d}a\,\mathrm{d}s\right).
\end{aligned}$$

We treat the product $\pi\mu_{o_j}$ of the policy $\pi$ and the state distribution $\mu_{o_j}$ as one input variable to the Lagrangian. This treatment makes sense because we want to solve for this product, meaning we are not interested in the value of $\pi$ or $\mu_{o_j}$ independently.

We can simplify this Lagrangian to a smaller more workable version

$$
\begin{aligned}
L(\pi\mu_{o_j}, \eta_{o_j}, \xi_{o_j}, \zeta_{o_j}, \theta_{o_j}, \lambda_{o_j}) = \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s) \Bigg[ R_{sa} - \eta_{o_j} \log\left(\frac{\pi(a|s,o_j)\mu_{o_j}(s)}{q(a,s|o_j)}\right) + q(o_j)\xi_{o_j}\log(\hat{p}(o_j|s,a)) \\
+ q(o_j)\sum_o \Big(\zeta_{o_j}(a,s,o)\hat{p}(o|s,a)\Big) - q(o_j)\zeta_{o_j}(a,s,o_j) \\
- \theta_{o_j}^T\phi(s) - \lambda_{o_j}\Bigg] + \eta_{o_j}\varepsilon_{o_j} + \xi_{o_j}\kappa + \theta_{o_j}^T\hat{\phi}_{o_j} + \lambda_{o_j} \\
+ \iint_{a,s} \xi_{o_j} \sum_{o\neq o_j} q(a,s,o)\log\hat{p}(o|s,a)\,\mathrm{d}a\,\mathrm{d}s \\
+ \iint_{a,s} \sum_o \zeta_{o_j}(a,s,o)\hat{p}(o|s,a)\left(\sum_{o_k\neq o_j} q(a,s,o_k)\right) \\
- \sum_{o\neq o_j} \zeta_{o_j}(a,s,o)q(a,s,o)\,\mathrm{d}a\,\mathrm{d}s \\
= \iint_{a,s} \pi(a|s,o_j)\mu_{o_j}(s)\left[R_{sa} - \eta_{o_j}\log\left(\frac{\pi(a|s,o_j)\mu_{o_j}(s)}{q(a,s|o_j)}\right) + A_{o_j}(a,s) - \theta_{o_j}^T\phi(s) - \lambda_{o_j}\right] \\
+ \eta_{o_j}\varepsilon_{o_j} + \xi_{o_j}\kappa + \theta_{o_j}^T\hat{\phi}_{o_j} + \lambda_{o_j} + B_{o_j}(a,s)
\end{aligned}
$$

with:

$$
\begin{aligned}
A_{o_j}(a,s) &= q(o_j)\left(\xi_{o_j}\log(\hat{p}(o_j|s,a)) + \sum_o\Big[\zeta_{o_j}(a,s,o)\hat{p}(o|s,a)\Big] - \zeta_{o_j}(a,s,o_j)\right), \\
B_{o_j}(a,s) &= \iint_{a,s}\sum_{o\neq o_j} q(a,s,o)\left(\eta_{o_j}\log\hat{p}(o|s,a) + \sum_{o_k}\Big[\zeta_{o_j}(a,s,o_k)\hat{p}(o_k|s,a)\Big] - \zeta_{o_j}(a,s,o)\right)\mathrm{d}a\,\mathrm{d}s.
\end{aligned}
$$

Note that $B_{o_j}$ is independent of $\pi\mu_{o_j}$.

The simpler Lagrangian allows us to easily take the derivative w.r.t. $\pi(a|s,o_j)\mu_{o_j}(s)$ which is

$$
\frac{\partial}{\partial\,\pi(a|s,o_j)\mu_{o_j}(s)} L(\pi\mu_{o_j}, \eta_{o_j}, \xi_{o_j}, \zeta_{o_j}, \theta_{o_j}, \lambda_{o_j}) = R_{sa} - \eta_{o_j}\log\left(\frac{\pi(a|s,o_j)\mu_{o_j}(s)}{q(a,s|o_j)}\right) + A_{o_j}(a,s) - \theta_{o_j}^T\phi(s) - \lambda_{o_j} - \eta_{o_j}.
$$

By setting the gradient of the Lagrangian w.r.t. $\pi(a|s,o_j)\mu_{o_j}(s)$ to zero we obtain the closed form solution

$$
\pi(a|s,o_j)\mu_{o_j}(s) = q(a,s|o_j)\exp\left(\frac{R_{sa} - \theta_{o_j}^T\phi(s) + A_{o_j}(a,s)}{\eta_{o_j}}\right)\exp\left(-\frac{\lambda_{o_j} + \eta_{o_j}}{\eta_{o_j}}\right).
$$

If we insert $\pi(a|s,o_j)\mu_{o_j}(s)$ in the probability constraint, we can solve for $\exp\left(-\frac{\lambda_{o_j}+\eta_{o_j}}{\eta_{o_j}}\right)$, so that we have

$$1 = \iint_{a,s} q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+A_{o_j}(a,s)}{\eta_{o_j}}\right)\exp\left(-\frac{\lambda_{o_j}+\eta_{o_j}}{\eta_{o_j}}\right).\,\mathrm{d}a\,\mathrm{d}s$$

$$\Leftrightarrow \quad \left[\iint_{a,s} q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+A_{o_j}(a,s)}{\eta_{o_j}}\right)\mathrm{d}a\,\mathrm{d}s\right]^{-1} = \exp\left(-\frac{\lambda_{o_j}+\eta_{o_j}}{\eta_{o_j}}\right).$$

We can now insert $\pi(a|s,o_j)\mu_{o_j}(s)$ into the Lagrangian to obtain the dual function

$$\begin{aligned} g(\eta_{o_j},\xi_{o_j},\zeta_{o_j},\theta_{o_j},\lambda_{o_j}) = \iint_{a,s}\pi(a|s,o_j)\mu_{o_j}(s)\Bigg[R_{sa}-\eta_{o_j}\log\left(\frac{q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+A_{o_j}(a,s)}{\eta_{o_j}}\right)\exp\left(-\frac{\lambda_{o_j}+\eta_{o_j}}{\eta_{o_j}}\right)}{q(a,s|o_j)}\right) \\ +A_{o_j}(a,s)-\theta_{o_j}^T\phi(s)-\lambda_{o_j}\Bigg]\,\mathrm{d}a\,\mathrm{d}s+\eta_{o_j}\varepsilon_{o_j}+\xi_{o_j}\kappa+\theta_{o_j}^T\hat{\phi}_{o_j}+\lambda_{o_j}+B_{o_j}(a,s) \\ = \iint_{a,s}\pi(a|s,o_j)\mu_{o_j}(s)\left[\log\left(\exp\left(-\frac{\lambda_{o_j}+\eta_{o_j}}{\eta_{o_j}}\right)\right)-\lambda_{o_j}\right]\mathrm{d}a\,\mathrm{d}s+\eta_{o_j}\varepsilon_{o_j}+\xi_{o_j}\kappa+\theta_{o_j}^T\hat{\phi}_{o_j}+\lambda_{o_j} \\ +B_{o_j}(a,s). \end{aligned}$$

At this point we use the probability constraint to remove $\iint_{a,s}\pi(a|s,o_j)\mu_{o_j}(s)\,\mathrm{d}a\,\mathrm{d}s$ and we insert the solution for the expression $\exp\left(-\frac{\lambda_{o_j}+\eta_{o_j}}{\eta_{o_j}}\right)$, which gives us

$$\begin{aligned} g(\eta_{o_j},\xi_{o_j},\zeta_{o_j},\theta_{o_j},\lambda_{o_j}) =& -\eta_{o_j}\log\left(\left[\iint_{a,s} q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+A_{o_j}(a,s)}{\eta_{o_j}}\right)\mathrm{d}a\,\mathrm{d}s\right]^{-1}\right) \\ &-\lambda_{o_j}+\eta_{o_j}\varepsilon_{o_j}+\xi_{o_j}\kappa+\theta_{o_j}^T\hat{\phi}_{o_j}+\lambda_{o_j}+B_{o_j}(a,s) \\ =&\eta_{o_j}\log\left(\iint_{a,s} q(a,s|o_j)\exp\left(\frac{R_{sa}-\theta_{o_j}^T\phi(s)+A_{o_j}(a,s)}{\eta_{o_j}}\right)\mathrm{d}a\,\mathrm{d}s\right) \\ &+\eta_{o_j}\varepsilon_{o_j}+\xi_{o_j}\kappa+\theta_{o_j}^T\hat{\phi}_{o_j}+B_{o_j}(a,s). \end{aligned}$$

The parameter $\lambda$ cancels out and we can approximate the integral and sum inside the logarithm. To approximate them, we notice that they can be seen as an expectation w.r.t. the probability $q(a,s)$. We have access to samples from $q(a,s)$, so by using these we approximate the expectation with the average over the samples $(a^{[i]},s^{[i]})$ to get

$$\begin{aligned} g(\eta_{o_j},\xi_{o_j},\zeta_{o_j},\theta_{o_j}) \approx \eta_{o_j}\log\left(\sum_{i=1}^{N}\frac{q(a^{[i]},s^{[i]}|o_j)}{q(a^{[i]},s^{[i]})}\exp\left(\frac{R_{sa}^{[i]}-\theta_{o_j}^T\phi(s^{[i]})+A_{o_j}(a^{[i]},s^{[i]})}{\eta_{o_j}}\right)\mathrm{d}a\,\mathrm{d}s\right) \\ +\eta_{o_j}\varepsilon_{o_j}+\xi_{o_j}\kappa+\theta_{o_j}^T\hat{\phi}_{o_j} \\ +\sum_{i=1}^{N}\sum_{o\neq o_j}\frac{q(a^{[i]},s^{[i]}|o)}{q(a^{[i]},s^{[i]})}q(o)\Big(\eta_{o_j}\log\hat{p}(o|s^{[i]},a^{[i]})+\sum_{o_k}\Big[\zeta_{o_j}(a^{[i]},s^{[i]},o_k)\hat{p}(o_k|s^{[i]},a^{[i]})\Big] \\ -\zeta_{o_j}(a^{[i]},s^{[i]},o)\Big). \end{aligned}$$

The variable $N$ is the number of samples. To solve the original optimization problem we minimize the dual function $g(\eta_{o_j}, \xi_{o_j}, \zeta_{o_j}, \theta_{o_j})$ such that $\eta_{o_j} > 0$ and $\xi_{o_j} > 0$ [6].
The dual function can be computed without knowledge of $q(s)$ by using the equation

$$\frac{q(a,s|o)}{q(a,s)} = \frac{q(o|s,a)}{q(o)}.$$

We can optimize the dual function using a numeric solver. To use the solver efficiently, we can compute the gradients of the Lagrangian multipliers. Although, they will only be accurate for the EM variant. For the non-EM variant they are not because we assume that $\hat{p}(o|s,a)$ is independent of the Lagrangian multipliers, which is not true. That said we used these gradients for the non-EM variant of our approach in the experiments and at least for the Reaching Task there was no major difference between the performance of the EM and non-EM variant.

The gradients are

$$\frac{\partial g}{\partial \theta_{o_j}} = \hat{\phi}_{o_j} - \frac{\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]}) \left(\phi(s^{[i]})\right)}{\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]})},$$

$$\frac{\partial g}{\partial \eta_{o_j}} \approx \varepsilon_{o_j} + \log\left(\frac{1}{N}\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]})\right) - \frac{\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]}) \left(\frac{R_{sa}^{[i]} - \theta_{o_j}^T \phi(s^{[i]}) + A_{o_j}(a^{[i]}, s^{[i]})}{\eta}\right)}{\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]})},$$

$$\frac{\partial g}{\partial \xi_{o_j}} \approx \kappa + \frac{\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]}) \left(q(o_j) \log \hat{p}(o_j|s^{[i]}, a^{[i]})\right)}{\sum_{i=1}^{N} C_{o_j}(a^{[i]}, s^{[i]})} + \frac{1}{N}\sum_{i=1}^{N}\sum_{o \neq o_j} \frac{q(o|s^{[i]}, a^{[i]})}{q(o)} q(o) \log \hat{p}(o|s^{[i]}, a^{[i]}),$$

with

$$C_{o_j}(a,s) = \frac{q(o|s,a)}{q(o)} \exp\left(\frac{R_{sa} - \theta_{o_j}^T \phi(s) + A_{o_j}(a,s)}{\eta_{o_j}}\right).$$

We do not need a gradient for $\zeta_{o_j}$ as we are using a different way to update it.

### A.3 Derivation of the Update Rule

We are taking the derivate of the Lagrangian w.r.t $\hat{p}(o|s,a)$ and solve for $\zeta_{o_j}$ to get our odd update rule. The derivation is

$$\begin{aligned}\frac{\partial L}{\partial \hat{p}(o|s,a)} = {} & \frac{\partial}{\partial \hat{p}(o|s,a)} \xi_{o_j} \left(\kappa + \iint_{a,s} p(s,a) \sum_{o} p(o|s,a) \log \hat{p}(o|s,a) \, \mathrm{d}a \, \mathrm{d}s\right) \\ & + \frac{\partial}{\partial \hat{p}(o|s,a)} \iint_{a,s} \sum_{o} \zeta_{o_j}(a,s,o) \left(\sum_{o_k} [p(a,s,o_k)] \hat{p}(o|s,a) - p(a,s,o) \, \mathrm{d}a \, \mathrm{d}s\right) \\ = {} & \frac{\xi_{o_j} p(a,s,o)}{\hat{p}(o|s,a)} + \zeta_{o_j}(a,s,o) p(a,s).\end{aligned}$$

Setting the derivative to zero gives us the equation we have used to in the option optimization section ,i.e.,

$$\zeta_{o_j}(a,s,o) = -\frac{\xi_{o_j} p(o|s,a)}{\hat{p}(o|s,a)}.$$

## A.4 Derivation of the Lower Bound

We want to show that we get a lower bound for our Lagrangian by approximating $p(o|s,a)$ inside the logarithm. Remember that for the EM-approach we do not need the constraint $p(a,s,o) = \hat{p}(o|s,a)p(a,s)$, so the $p(o|s,a)$ inside the logarithm is the only $p(o|s,a)$ we want to replace with a fixed value.
To show that we have a lower bound for the Lagrangian by fixing $p(o|s,a)$ it is enough to show that fixing it gives us a lower for only the part of the Lagrangian, which we change. The relevant part we are referring to is

$$\xi_{o_j}\left(\kappa + \iint_{a,s} p(s,a)\sum_o p(o|s,a)\log p(o|s,a)\,\mathrm{d}a\,\mathrm{d}s\right).$$

To this expression we can add the term

$$-\xi\iint_{a,s} p(a,s)D_{\mathrm{KL}}(p(o|s,a)||\tilde{p}(o|s,a))\,\mathrm{d}a\,\mathrm{d}s.$$

This term is always negative or zero, because the KL-divergence is always greater or equal to zero. Adding it to the Lagrangian gives us a lower bound, which we can see here:

$$\begin{aligned}
&\xi\left(\kappa + \iint_{a,s} p(s,a)\sum_o p(o|s,a)\log p(o|s,a)\,\mathrm{d}a\,\mathrm{d}s\right) - \xi\iint_{a,s} p(a,s)\sum_o p(o|s,a)\log\left(\frac{p(o|s,a)}{\tilde{p}(o|s,a)}\right)\\
&\quad= \xi\left(\kappa + \iint_{a,s} p(s,a)\sum_o p(o|s,a)\left(\log p(o|s,a) + \log\left(\frac{\tilde{p}(o|s,a)}{p(o|s,a)}\right)\right)\,\mathrm{d}a\,\mathrm{d}s\right)\\
&\quad= \xi\left(\kappa + \iint_{a,s} p(s,a)\sum_o p(o|s,a)\log\tilde{p}(o|s,a) + \,\mathrm{d}a\,\mathrm{d}s\right)
\end{aligned}$$

This equation shows that replacing $p(o|s,a)$ with a fixed value $\tilde{p}(o|s,a)$ (that still fulfills the requirements of a probability) creates a lower bound for the Lagrangian.

## A.5 Gating Optimization

The constraint optimization problem for the gating is given by

$$\begin{aligned}
&\max_{\pi(o|s)} \int_s \sum_o \pi(o|s)\mu(s)R_{so}\ \mathrm{d}s, \\
\text{s.t:}\ \ &\varepsilon \geq \mathrm{E}_s[D_{\mathrm{KL}}(\pi(o|s)||q(o|s))], \\
&\alpha \leq \mathrm{E}_s[\mathrm{H}_\pi[o|s]], \\
&1 = \int_s \sum_o \pi(o|s)\mu(s)\ \mathrm{d}s.
\end{aligned}$$

We can compute the Lagrangian of this optimization problem, which is

$$\begin{aligned}
L(\pi,\eta,\beta,\lambda) = &\int_s \sum_o \pi(o|s)\mu(s)R_{so}\ \mathrm{d}s \\
&+ \eta\left(\varepsilon - \int_s \sum_o \pi(o|s)\mu(s)\log\left(\frac{\pi(o|s)}{q(o|s)}\right)\right) \\
&+ \beta\left(-\alpha - \int_s \sum_o \pi(o|s)\mu(s)\log(\pi(o|s))\ \mathrm{d}s\right) \\
&+ \lambda\left(1 - \int_s \sum_o \pi(o|s)\mu(s)\ \mathrm{d}s\right)
\end{aligned}$$

$$\begin{aligned}
&= \int_s \sum_o \pi(o|s)\mu(s)\left[R_{so} - \eta\log\left(\frac{\pi(o|s)}{q(o|s)}\right) - \beta\log(\pi(o|s)) - \lambda\right]\mathrm{d}s + \eta\varepsilon - \alpha\beta + \lambda \\
&= \int_s \sum_o \pi(o|s)\mu(s)\left[R_{so} - \eta\log\left(\frac{\pi(o|s)^{\frac{\eta+\beta}{\eta}}}{q(o|s)}\right) - \lambda\right]\mathrm{d}s + \eta\varepsilon - \alpha\beta + \lambda.
\end{aligned}$$

Taking the derivative of the Lagrangian w.r.t. $\pi$ gives us

$$\frac{\partial}{\partial\pi}L(\pi,\eta,\beta,\lambda) = \mu(s)\left[R_{so} - \eta\log\left(\frac{\pi(o|s)^{\frac{\eta+\beta}{\eta}}}{q(o|s)}\right) - \lambda - \eta - \beta\right].$$

Setting the derivative to zero gives us a closed form solution for $\pi(o|s)$, i.e.,

$$\pi(o|s) = q(o|s)^{\frac{\eta}{\eta+\beta}}\exp\left(\frac{R_{so}}{\eta+\beta}\right)\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right).$$

If we insert $\pi(o|s)$ in the probability constraint, we can solve for $\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right)$ to get

$$\begin{aligned}
&1 = \int_s \sum_o q(o|s)^{\frac{\eta}{\eta+\beta}}\exp\left(\frac{R_{so}}{\eta+\beta}\right)\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right)\mu(s)\ \mathrm{d}s \\
\Leftrightarrow\quad &\left[\int_s \sum_o q(o|s)^{\frac{\eta}{\eta+\beta}}\mu(s)\exp\left(\frac{R_{so}}{\eta+\beta}\right)\mathrm{d}s\right]^{-1} = \exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right).
\end{aligned}$$

We can now insert $\pi(o|s)$ into the Lagrangian to obtain the dual function, which is

$$\begin{aligned}
g(\eta,\beta,\lambda) &= \int_s \sum_o \pi(o|s)\mu(s)\left[R_{so} - \eta\log\left(\frac{\left(q(o|s)^{\frac{\eta}{\eta+\beta}}\exp\left(\frac{R_{so}}{\eta+\beta}\right)\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right)\right)^{\frac{\eta+\beta}{\eta}}}{q(o|s)}\right) - \lambda\right]\,\mathrm{d}s + \eta\varepsilon - \alpha\beta + \lambda \\
&= \int_s \sum_o \pi(o|s)\mu(s)\left[R_{so} - \eta\log\left(\frac{R_{so}}{\eta}\right) - \eta\log\left(\left(\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right)\right)^{\frac{\eta+\beta}{\eta}}\right) - \lambda\right]\,\mathrm{d}s + \eta\varepsilon - \alpha\beta + \lambda \\
&= \int_s \sum_o \pi(o|s)\mu(s)\,\mathrm{d}s\left[-(\eta+\beta)\log\left(\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right)\right) - \lambda\right] + \eta\varepsilon - \alpha\beta + \lambda.
\end{aligned}$$

At this point we can use the probability constraint to remove $\int_s \sum_o \pi(o|s)\mu(s)\,\mathrm{d}s$ and we can insert $\exp\left(-\frac{\lambda+\eta+\beta}{\eta+\beta}\right)$, which we calculated earlier to get

$$\begin{aligned}
g(\eta,\beta,\lambda) &= -(\eta+\beta)\log\left(\left[\int_s \sum_o q(o|s)^{\frac{\eta}{\eta+\beta}}\mu(s)\exp\left(\frac{R_{so}}{\eta+\beta}\right)\,\mathrm{d}s\right]^{-1}\right) - \lambda + \eta\varepsilon - \alpha\beta + \lambda \\
&= (\eta+\beta)\log\left(\int_s \sum_o q(o,s)q(o|s)^{\left(\frac{\eta}{\eta+\beta}-1\right)}\exp\left(\frac{R_{so}}{\eta+\beta}\right)\,\mathrm{d}s\right) + \eta\varepsilon - \alpha\beta.
\end{aligned}$$

The parameter $\lambda$ cancels out and we can approximate the integral and sum inside the logarithm. To approximate them, we notice that they can be seen as an expectation w.r.t. the probability $q(o,s)$. We have access to samples from $q(o,s)$, so by using these we can approximate the expectation with the average over the samples $(o^{[i]},s^{[i]})$. This approximation gives us

$$g(\eta,\beta) \approx (\eta+\beta)\log\left(\frac{1}{N}\sum_{i=1}^{N} q(o^{[i]}|s^{[i]})^{\frac{-\beta}{\eta+\beta}}\exp\left(\frac{R_{so}^{[i]}}{\eta+\beta}\right)\right) + \eta\varepsilon - \alpha\beta.$$

The variable $N$ is the number of samples. We solve this dual function for $\eta > 0$ and $\beta > 0$ [6] to get the Lagrangian multipliers for the closed form solution.

The dual function can be optimized numerically. Using gradients the optimization can be sped up. The gradients are

$$\frac{\partial g(\eta,\beta)}{\partial \eta} \approx \varepsilon + \log\left(\frac{1}{N}\sum_{i=1}^{N} X(s^{[i]},o^{[i]})\right) - \frac{\sum_{i=1}^{N} X(s^{[i]},o^{[i]})\left(\frac{R_{so}^{[i]} - \beta\log q(o^{[i]}|s^{[i]})}{\eta+\beta}\right)}{\sum_{i=1}^{N} X(s^{[i]},o^{[i]})},$$

$$\frac{\partial g(\eta,\beta)}{\partial \beta} \approx -\alpha + \log\left(\frac{1}{N}\sum_{i=1}^{N} X(s^{[i]},o^{[i]})\right) - \frac{\sum_{i=1}^{N} X(s^{[i]},o^{[i]})\left(\frac{R_{so}^{[i]} + \eta\log q(o^{[i]}|s^{[i]})}{\eta+\beta}\right)}{\sum_{i=1}^{N} X(s^{[i]},o^{[i]})},$$

with

$$X(s^{[i]},o^{[i]}) = \exp\left(\frac{R_{so}^{[i]} - \beta\log q(o^{[i]}|s^{[i]})}{\eta+\beta}\right).$$