# Hierarchical Reinforcement Learning with Parameters *

**Maciej Klimek**
deepsense.ai

**Piotr Miłoś**
University of Warsaw
deepsense.ai

**Henryk Michalewski**
Institute of Mathematics of the Polish Academy of Sciences
deepsense.ai

**Abstract:** In this work we introduce and evaluate a model of Hierarchical Reinforcement Learning with Parameters. In the first stage we train agents to execute relatively simple actions like reaching or gripping. In the second stage we train a hierarchical manager to compose these actions to solve more complicated tasks. The manager may pass parameters to agents thus controlling details of undertaken actions. The hierarchical approach with parameters can be used with any optimization algorithm.

In this work we adapt to our setting methods described in [1]. We show that their theoretical foundation, including monotonicity of improvements, still holds. We experimentally compare the hierarchical reinforcement learning with the standard, non-hierarchical approach and conclude that the hierarchical learning with parameters is a viable way to improve final results and stability of learning.

**Keywords:** Hierarchical learning, Learning in simulation, Grasping, Trust Region Policy Optimization

## 1 Introduction

The field of robotic control has witnessed striking advances in recent years, many of them due to applications of reinforcement learning and deep learning techniques [2, 3, 4, 5, 6, 1]. Nevertheless a long term goal of learning complicated tasks with sparse rewards still seems elusive. So far it requires a substantial human intervention or prolonged learning on a very large number of samples. In this work we attempt to achieve difficult objectives dividing a complex problem into subproblems with a Manager deciding when a given subproblem should be solved. The hierarchical approach to reinforcement learning has been considered in foundational works [7, 8] and recently advanced in [9, 10]. A similar idea of a hierarchical decomposition of complicated tasks was also used in the context of optimal control [11, 12].

In the field of robotics it seems to be particularly justified to define a set of elementary actions used for more complicated tasks. For instance, the **reach-and-move** task (see Figure 2 and videos) considered in Sections 1.3 and 4 divides into two simpler actions of reaching of an object and moving it to a target destination. Ultimately, a library of reusable agents performing basic actions can be utilized in new complex tasks. We argue that challenging reinforcement learning problems often have a natural hierarchical structure. For instance one can attempt to solve a difficult Atari 2600 game considered in [13], combining an exploratory strategy forcing the agent to visit more rooms in the maze with a strategy focused on maximizing rewards. In [10] an agent learns basics of locomotion as a part of a more complicated navigational task. In [5] is considered a task of moving a LEGO brick which can be divided into a task of gripping of a brick and moving it to another location.

*All authors contributed equally to this work.

1st Conference on Robot Learning (CoRL 2017), Mountain View, United States.

As shown by experiments presented in Section 4, combining hierarchical learning with passing of parameters can potentially lead to better utilization of resources and as a result to more expressive, sample-efficient and easier to train models. In Section 3 we present mathematical foundations of our work. In Section 2 we describe other methods which can be used to solve control tasks considered in this work and in particular in Section 2 we discuss relation of Hierarchical Reinforcement Learning with Parameters to other hierarchical models present in the literature. In the rest of the introduction we define the concept of Hierarchical Reinforcement Learning with Parameters and briefly discuss mathematical foundations of our work and the experimental setting.

### 1.1 Definition of Hierarchical Reinforcement Learning with Parameters (HRLP)

The main contribution of this papers consists in combining of hierarchical learning with parametrized tasks with a modern reinforcement learning algorithm and with deep learning. Hierarchical reinforcement learning with parameters (HRLP) consists of the following components (see Figure 1, left): (1) the Manager, which selects one of $N$ available macro-actions and passes a vector of parameters to the selected macro-action; (2) the macro-action executes a sequence of lower-level control actions, such as changing of a torque of a selected actuator; (3) after the selected macro-action is executed, the Manager observes updated information about the state of the Environment and can pass this information back to macro-actions. In experiments in Section 4 the full information is passed to macro-actions, but in general part of it can be hidden from some macro-actions.

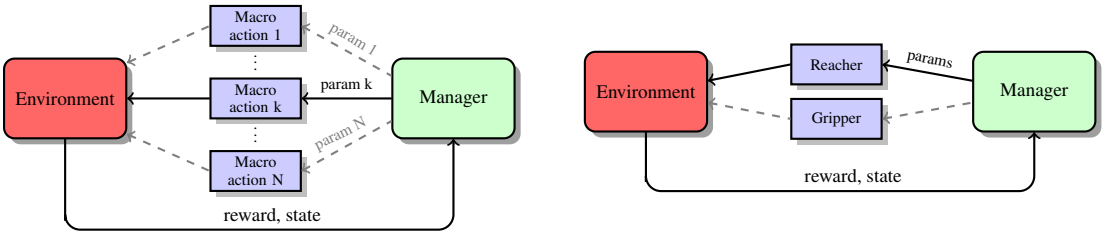


*Figure 1: An abstract version of HRLP (left) and an example of HRLP for* **reach-and-grip** *task.*

In training of HRLP models we first train all macro-actions and then train the Manager, assuming that models of macro-actions are fixed. In some of our experiments we also allow a special macro-action which gives the Manager a direct access to all control actions. That is, in principle we allow a possibility that the Manager can resort to micro-management of control actions, such as adjustments of torques, if in the process of training it appears that macro-actions are not sufficiently effective. Presence and absence of this micro-management facility is explored in experiments in Section 4. Figure 1 (right) shows components of HRLP in the case of the **reach-and-grip** experiment.

### 1.2 Mathematical foundations of HRLP

HRLP has solid mathematical foundations when combined with optimization procedure introduced in [1]. The underlying algorithm in [1] derived from [14] guarantees monotonic improvements in the process of policy optimization. In Theorem 1 in Section 3.2 we show that this guarantee extends to the hierarchical setting. In Corollary 1 we also provide a lower bound on a policy improvement. In our view the bound may be easier to interpret than the one originally proposed in [1]. The bound in Corollary 1 is expressed solely in terms of the distance between the current and updated policies. Corollary 1 is applicable both to the hierarchical and non-hierarchical tasks.

### 1.3 Experiments with HRLP

As a testbed for HRLP we adopted **reach-and-grip** and **reach-and-move** tasks simulated in the physics engine MuJoCo [15] embedded inside of an OpenAIGym [16] environment (see Figure 3 and videos). Tasks **reach-and-grip** and **reach-and-move** are divided into subproblems of reaching and gripping and respectively of reaching and moving. The Gripper, Manager, Mover and Reacher take as input positions and velocities of joints. The Gripper, Mover and Reacher output torques applied to actuators. The Gripper, Mover and Reacher are trained beforehand, see Section 4.1 for description of these experiments. The Reacher and Mover are trained so that they can reach various locations of the target. In Sections 4.2 and 4.3 we describe a number of hierarchical and analogous non-hierarchical experiments for the **reach-and-move** and **reach-and-grip** tasks respectively.

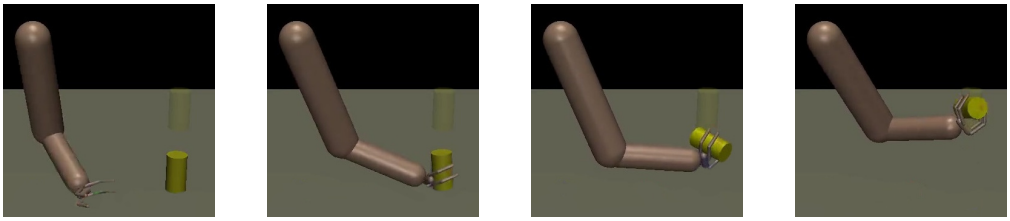

*Figure 2: Stages of the* **reach-and-move** *task (from the left): (1) randomized starting position, (2) reaching the capsule, (3-4) attempting to move the capsule to the designated target position which is visible as a shadow capsule.*

Virtually any model-free RL learning algorithm can be adapted to HRLP. In experiments we use the Trust Region Policy Optimization (TRPO) algorithm [1], which proved to be highly effective on a number of robotic benchmarks [17]. TRPO is designed as a model-free reinforcement learning algorithm. Its applicability, as well as the applicability of the hierarchical TRPO implemented in this work, is not limited to robotic tasks. An expected downside of this generality is a relative poor sample efficiency and difficulties in achieving long-term goals (see [18, Section 6] for a broad comparison of model-free algorithms involving trust region policy optimization). At the current stage of our experiments it would be difficult to provide a complete assessment of sample efficiency of our method, however preliminary results suggest that training of the Manager requires up to ten times less samples then training of the agent solving the complete task. Hence we believe that our hierarchical approach can partially address request 2 on page 85 of [6] and an OpenAI request for research [19] and overall be a step towards resolving both the problem of sample efficiency and the problem of achieving long-term goals.

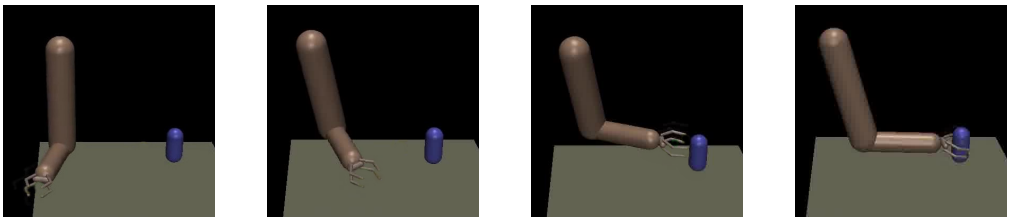

*Figure 3: Stages of the* **reach-and-grip** *task (from the left): 1) randomized starting position, (2) reaching the capsule, (3-4) attempting to grip the capsule.*

## 2 Related work

**Hierarchical reinforcement learning.**Temporally extended macro-actions were introduced by Sutton et al. in [7]. This framework does not directly describe our setting yet is general enough to encompass it. Namely, a parameterized macro-action can be considered as an infinite set of options indexed by parameters, the manager is then a policy over options and micro-actions are primitive actions.

The paper [20] pinpoints important conceptual advantages of explicit parameterization. The authors introduced parametrized skills and indicated that a parameter space having some structure enables generalization. During training an agent can synthesize its knowledge about a parametrized task and later on-demand solve the task for previously unseen parameters.

In our work we use pretrained macro-actions. This in particular requires that in advance we define their intended behavior. There has been several attempts to discover macro-actions automatically, for example [21, 22, 23]. We note that parameterizations are also considered in [22, 23].

**Relation to [24, 12].** Papers [24, 12] consider the hierarchical setting which on a high level is equivalent to ours. They differ on the algorithmic level. In [12] the authors use methods of optimal control. Two model-free algorithms are developed in [24]. The authors of [24] use two approximators, one to choose a macro-action and another determining parameters of its execution. Their Q-PAMDP($k$) learning algorithm alternately updates the parameters of one of the approximators keeping the other fixed (they experiment also with a joint optimization procedure, but obtain weaker results). In our experiments we use a single deep neural network depicted in Figure 9 in Appendix

B and joint optimization. The results of [24] are in line with our findings that hierarchical approach is beneficial.

**Relation to [9].** Instead of dividing a complicated task into simpler subproblems, authors of [9] propose a general scheme in which simpler skills are discovered and acquired in pre-training and then re-used in actual training. Simpler skills are not specified upfront, but it is required to define a reward function which will 'encourage learning of simpler skills. That is, instead of devising a hierarchy of subtasks, authors of [9] propose focusing on a hierarchy of reward functions. The Manager used in [9] does not pass parameters to the agent performing lower level tasks. Experiments presented in [9] are based on an appropriately modified TRPO algorithm.

**Relation to [10].** Authors of [10] explicitly divide a more complicated task into a low-level task such as locomotion and a high-level task such as navigation. The high level agent passes a vector of parameters to the low-level agent. The low-level agent a priori is not specialized in one specific task. Experiments presented in [10] are based on the actor-critic algorithm introduced in [25].

**The reach-and-grip and reach-and-move tasks (relation to [2, 5]).** In papers [2, 5] impressive results are achieved in tasks very similar to **reach-and-grip** and **reach-and-move** . Nevertheless, the hierarchical model presented in this work allows for a systematic learning of a spectrum of re-usable low level skills.

**Trust Region Policy Optimization as the core algorithm (relation to [6, 1]).** The dissertation [6] and paper [1] introduced the Trust Region Policy Optimization (TRPO) algorithm which we use in experiments presented in Section 4. The algorithm is characterized by a very good practical performance [17] and simultaneously offers good mathematical guarantees, which in the next Section in Theorem 1 we generalize to the hierarchical case.

## 3 Hierarchical Reinforcement Learning with Parameters and monotonic improvements

In the paragraph below we introduce basic notions related to Markov Decision Processes in the context of reinforcement learning. In Section 3.1 we introduce the hierarchical version of the Kullback-Leibler divergence and in Section 3.2 we show that monotonicity of improvements established in [6, 1] for the non-hierarchical scenario extends to the hierarchical case. The most important results in this Section are inequalities (2), (5) which rephrase results of [1, Section 3] in the context of HRLP and two new inequalities (4) and (6), which are analogous to (2) and (5) but arguably easier to interpret. All four inequalities are also valid in the non-hierarchical case.

**Markov Decision Processes.**Following [26, Section 3.6] we define a Markov Decision Process (MDP) as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a set of actions available to an agent, $P$ is the transition kernel, $r$ is the reward function, $\rho_0$ is the distribution of the initial state, and $\gamma \in (0, 1)$ is the discount factor. We assume that $\mathcal{S}, \mathcal{A}$ are finite and in particular rewards are bounded by a number $R < +\infty$, but definitions and results below hold also in the continuous setting.

Given a stochastic policy $\pi$, which for a given state assigns probability distribution over possible actions, we define the action-value function $Q_\pi$, the value function $V_\pi$ and the advantage function $A_\pi$ using the following formulas: $Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{+\infty}\gamma^t r_{t+1}|s_0 = s, a_0 = a\right]$, $V_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{+\infty}\gamma^t r_{t+1}|s_0 = s\right]$, $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$. The symbol $\mathbb{E}_\pi$ refers to the fact that the trajectory, and thus also rewards $\{r_t\}$, are generated by an agent following the policy $\pi$. As a shortcut we use $\eta(\pi)$ for the value function, when the initial state is sampled according to $\rho_0$. We define also $\rho_\pi$ the discounted visitation frequency[2] $\rho_\pi(s) = \sum_{t=0}^{\infty}\gamma^t \mathbb{P}_\pi(s_t = s) = \mathbb{E}_\pi\left[\sum_{t=0}^{+\infty}\gamma^t 1_{s_t=s}\right]$. Further let $L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s\in\mathcal{S}}\rho_\pi(s)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|s)A_\pi(s,a)$, with $\eta(\pi) = \mathbb{E}V_\pi(s_0)$, where $s_0$ is the initial state (possibly random). Following [1, Formulas (2,3)], we take $L_\pi(\tilde{\pi})$ as a proxy for $\eta(\tilde{\pi})$ and instead of directly solving the maximization problem for $\eta$, we solve a problem for $L_\pi$. Theorem 1 in [1] for the non-hierarchical learning and Theorem 1 in this work for hierarchical learning guarantee that if $\pi$ and $\tilde{\pi}$ are close enough, then through improving $L_\pi(\tilde{\pi})$ we also improve $\eta(\tilde{\pi})$.

[2]In the continuous case we would define $\rho_\pi(s) = \sum_{t=0}^{\infty}\gamma^t g_\pi(s)$, where $g_\pi(s)$ is the density of the state $s$ at a time $t$ under policy $\pi$. Then the definition of $L_\pi(\tilde{\pi})$ would use integrals instead of sums.

### 3.1 Hierarchical action space

Let $n \geq 1$ be the number of hierarchical macro-actions. Each takes as an argument a vector from the space $\mathcal{A}_k = \mathbb{R}^{d_k}$ (note that $d_k$ might be 0 in the case when a given macro-action is parameterless). We define the parameters space $\mathcal{A} = \bigsqcup_{k=1}^{n} \mathcal{A}_k$, where $\bigsqcup$ denotes the disjoint union. We will consider probability distributions on $\mathcal{A}$. Any such distribution $\mu$ decomposes naturally into $p = (p_1, \ldots, p_k)$, where $p$ a distribution on $\{1, \ldots, n\}$ and $\{\mu_k\}_{k=1}^{n}$, which are distributions on $\mathcal{A}_k$ respectively. In experiments in Section 4 $\mu_k$ are multivariate-Gaussian distributions. Assume that there are given distributions $\nu, \mu$ on a space $\mathcal{X}$. The total variation distance is defined as $D_{TV}(\nu, \mu) = \sup_B |\nu(B) - \mu(B)|$, where the supremum ranges over all measurable $B \subseteq \mathcal{X}$. If moreover $\nu$ is absolutely continuous with respect to $\mu$, following [27, 28], we define the Kullback–Leibler divergence as $D_{KL}(\nu, \mu) = \int_{\mathcal{X}} \log\left(\frac{d\nu}{d\mu}\right) d\nu$, where $\frac{d\nu}{d\mu}$ is the Radon-Nikodym derivative. The following Lemma provides a convenient characterization of the Kullback–Leibler divergence in the hierarchical setting.

**Lemma 1.** *Let $\mu, \tilde{\mu}$ be distributions on $\mathcal{A}$ such that $\mu$ is absolutely continuous with respect to $\tilde{\mu}$. Then*

$$D_{KL}(\mu, \tilde{\mu}) = D_{KL}(p, \tilde{p}) + \sum_{k=1}^{n} p_k D_{KL}(\mu_k, \tilde{\mu}_k). \tag{1}$$

Proof of the above Lemma is presented in Appendix A.

### 3.2 Monotonic improvement guarantee for hierarchical policies

As was explained at the end of the paragraph defining Markov Decision Processes, $L_\pi(\tilde{\pi})$ is a good approximation of $\eta(\tilde{\pi})$ in a neighborhood of the trajectory generated by $\pi$. This is formalized by the following lower bound for $\eta(\tilde{\pi})$ (for a proof sketch see Appendix A):

**Theorem 1.** *Let $\pi$ and $\tilde{\pi}$ be two $\mathcal{A}$–valued policies. Then*

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - \frac{2\epsilon\gamma}{(1-\gamma)^2} \left(D_{TV}^{\max}(\pi, \tilde{\pi})\right)^2, \tag{2}$$

*where $\epsilon = \sup_{s \in \mathcal{S}} |\mathbb{E}_{\tilde{\pi}(a|s)}[A^\pi(s, a)]|$ and $D_{TV}^{\max}(\pi, \tilde{\pi}) = \sup_{s \in \mathcal{S}} D_{TV}(\pi(\cdot|s), \tilde{\pi}(\cdot|s))$.*

The following Lemma is needed to derive the bounds in Corollary 1.

**Lemma 2.** *Let $\pi$ and $\tilde{\pi}$ be two policies. Then*

$$\max_{s \in \mathcal{S}} |\mathbb{E}_{\tilde{\pi}(a|s)}[A^\pi(s, a)]| \leq \frac{4R}{1-\gamma} D_{TV}^{\max}(\pi, \tilde{\pi}), \tag{3}$$

*where $R$ is the maximal reward.*

Proof of Lemma 2 is presented in Appendix A. From Theorem 1 and the above Lemma follows

**Corollary 1.** *Let $\pi$ and $\tilde{\pi}$ be two policies. Then*

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C_1 C_\gamma^1 R \left(D_{TV}^{\max}(\pi, \tilde{\pi})\right)^3, \quad C_1 = 8, C_\gamma^1 = \gamma/(1-\gamma)^3. \tag{4}$$

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C_2 C_\gamma^2 \epsilon \left(D_{KL}^{\max}\pi, \tilde{\pi}\right), \quad C_2 = 2, C_\gamma^2 = \gamma/(1-\gamma)^2. \tag{5}$$

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C_3 C_\gamma^3 R \left(D_{KL}^{\max}(\pi, \tilde{\pi})\right)^{3/2}, \quad C_3 = 2\sqrt{2}, C_\gamma^3 = \gamma/(1-\gamma)^3 \tag{6}$$

*where $D_{KL}^{\max}(\pi, \tilde{\pi}) = \sup_{s \in \mathcal{S}} D_{KL}(\pi(\cdot|s), \tilde{\pi}(\cdot|s))$.*

*Proof.* The proof follows by simple calculations and the Pinsker inequality [28, Theorem 1.4]: $D_{TV}(\nu, \mu) \leq \sqrt{\frac{D_{\mathrm{KL}}(\nu, \mu)}{2}}$, which holds for any distributions $\nu, \mu$ on the same space. □

*Remark.* Inequalities (2) and (5) are analogues of [1, (7) and (9)] and provide theoretical foundations of the TRPO algorithm for $\mathcal{A}$ valued polices. Our proof of these two inequalities follows the lines of the argument in [1, Section 3].

Additionally we also prove (4) and (6). Consider the right-hand sides. The dependence on the policies is expressed in terms of the Kullback-Leibler distance between $\pi$ and $\tilde{\pi}$. This is more explicit than (2) and (5) which also depends on $\epsilon$ defined in the formulation of Theorem 1, thus arguably easier to interpret and might be of theoretical interest.

# 4 Experimental results

Information about the model, optimization frameworks and physics engine are contained in Appendix B.1. In the following experiments $d$ is the Euclidean distance in $\mathbb{R}^3$, $H$ is the location of the hand (precisely a point where the center of the mass of the capsule would be if the capsule were properly gripped by the hand) and $C$ is the location of the center of mass of the capsule.

## 4.1 Learning of macro-actions

**Experiment 1. Gripper.** *Starting position*: we assume that the capsule is on a circle centered around the main axis of the arm. The palm is located close to the capsule. *Objective*: gripping the capsule. *Reward*: $c_1 d(H,C) + c_2 1_{d(H,C)<tr} + c_3 d(FT_1, FT_2)$, where $c_1, c_3 < 0$, $c_2 > 0$, $tr > 0$ is a certain threshold and $FT_1, FT_2$ are positions of the finger tips. In practice it is required to consider many "close" positions in order to make gripper usable in hierarchical experiments. To evaluate the Gripper we use

**Definition 1.** *(Sparse Rewards) We define a trajectory as successful if at least in one of its point both the conditions $d(H,C) < tr_1$ and $d(FT_1, FT_2) < tr_2$, are satisfied for certain thresholds $tr_1, tr_2 > 0$.*

**Experiment 2. Mover.** *Starting position:* we assume that the palm is close to the capsule. *Objective:* moving capsule to a designated place. *Reward:* A weighted sum of $-d(H,C)$ and $-d(C,T)$, where $T$ is the location of the designated target. We trained two versions of the Mover - MoverPlain and MoverNoised. MoverPlain is trained on pretty small set of starting positions. MoverNoised is trained on much bigger set of starting positions.

**Experiment 3. Reacher.** *Starting position:* We used various initial conditions. *Objective:* moving hand to the location of the capsule. *Reward:* $-d(H,C)$, sometimes we use additional reward to encourage some specific behavior.

## 4.2 Task reach-and-move

The reward used for this task is a weighted sum of $-d(H,C) - \texttt{TargetCoeff} \cdot d(C,T)$. To compare agents trained with different `TargetCoeff` we introduce $\texttt{TargetDistSum} = \sum_t -d(C,T)$.

**Experiment.** The position of the capsule is fixed. The designated target position of the capsule is fixed across all experiments and located above the capsule. The starting position of the arm is changing across episodes.

### 4.2.1 Non-hierarchical experiments

We managed to train a policy that successfully moves the capsule to the target position. In figure 4 we show results depending on the value of `TargetCoeff`. From movies (click to see video examples), we observed that agents with roughly $\texttt{TargetDistSum} \geq -23$ exhibit the proper reaching and moving behavior. Some agents with $\texttt{TargetDistSum} > -50$ learn to push the capsule towards the target position. Those with $\texttt{TargetDistSum} < -50$ managed only to reach the capsule. We conclude that it is hard to train a well-performing agent.

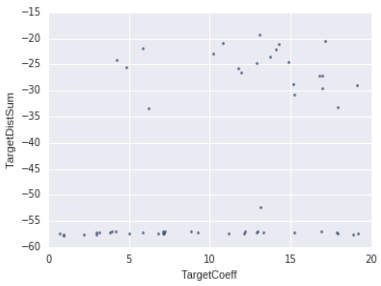


*Figure 4: Relationship between `TargetCoeff`and `TargetDistSum`. Each point indicates a single run.*

### 4.2.2 Hierarchical experiments

We managed to train a Manager using various Reacher and Mover macro-actions to successfully perform reaching and moving. The macro-actions are executed for 8 steps. The Manager passes a vector of 3 numbers to the Reacher. The vector is the desired position of the hand.

We tried three different Reacher macro-action policies. The first one, named the ReacherNormal, was trained with a capsule standing on the ground at the beginning of the episode. The ReacherStabilised was trained similarly but we included an addition reward to keep the forearm parallel to the ground. This facilitates its usage in the hierarchical experiments. The third, Reacher3D was trained with capsule placed randomly in $\mathbb{R}^3$. The best Reacher3D policy developed a strange behavior of

closing the palm while reaching, which made it difficult to use the agent in hierarchical setting. We use the Mover macro-actions as described in Section 4.1.

In Figure 5 we present our results. While results are worse compared to the non-hierarchical case, from the movies (click to see video examples) we observed that successful reaching and moving behavior is exhibited by agents with roughly $\texttt{TargetDistSum} > -45.0$.

| | ReacherNormal | ReacherStabilized | Reacher3D |
|---|---|---|---|
| MoverPlain | -52.778 | -43.552 | -56.989 |
| MoverNoised | -42.239 | -41.470 | -55.369 |

*Figure 5: Averaged values of* $\texttt{TargetDistSum}$ *for* 10 *runs.*

All agents trained using ReacherNormal and MoverNoised exhibited such successful behavior. This stands in sharp contrast with the non-hierarchical setting where substantial majority performed poorly. From Figure 5 we observe weak results for the Reacher3D. This agent tended to close the palm, thus the agent was hard to utilize together with the Mover, because the Mover was not trained in such conditions. The MoverPlain, trained on a small space of initial states, showed a similar incompatibility with the ReacherNormal and overall performed much weaker than the MoverNoised.

Describing our preferences using rewards can be hard ([29] describes a few interesting examples). Constraining the space of polices which the agent can learn might be another way of expressing the true objective. This can be realized using HRLP. In the constrained space of HRLP policies a good policy can have lower rewards but still exhibit desired behaviors. For example, as mentioned above, agents trained using HRLP with $\texttt{TargetDistSum} > -45.0$ exhibited successful moving behavior, whereas in the non-hierarchical case the threshold was around $-25.0$.

### 4.3 Task reach-and-grip

**Experiment 1** *Starting position*: we assume that the capsule is located in one position. The starting position of the palm is located randomly on the circle. *Objective*: gripping the capsule. *Reward*: The same as in the Gripper experiment in Section 4.1.

**Experiment 2** The position of the capsule and the starting position of the palm are located on a circle and changing independently across episodes.

Click here to see video examples illustrating Experiments 1,2.

#### 4.3.1 Hierarchical experiments

In this section to evaluate the results we use sparse rewards as described in Definition 1.

**Ad. Experiment 1.**Table 6 presents results of Experiment 1, when the Manager can use a Gripper and a Reacher. Experiments showed little dependency on other parameters, in particular on the random seed. On average every experiment required 1 million of samples, with the best agents being able to learn after 0.5 million samples. This accounts for 5 to 10 millions of interactions with the environment as the length of macro-actions is set to 10.

**Ad. Experiment 2.**Our results, summarized in the table in Figure 7, consist of four types of experiments differing by the macro-actions available for the Manager.[3] We managed to train a Manager using various Reacher and Mover macro-actions to successfully perform gripping. The macro-actions are executed for 15 steps. The Manager passes a vector of 3 numbers to the Reacher. The vector is the desired position of the hand. Our base gripper+reacher experiment achieved moderate results. We stress that they are still substantially better than their non-hierarchical counterparts in Section 4.3.2 (namely the ones with $\alpha \geq \pi/10^5$). We reckon that moderate results of HRLP stems from a problem already noticed in in Section 4.2.2 (and e.g. [9, Section 8]). Namely, when the Manager switches from one macro-action to another, the later might be in a region of the state-space not used during its pre-training. The first attempt to fix this issue was adding micro-actions, with the idea that the Manager can use "the direct control" to alleviate this problem. This indeed happened, though we observed that the Manager tends to abandon the usage

| | 0.03 | 0.06 | mean |
|---|---|---|---|
| 3 | 0.984 | 0.955 | 0.970 |
| 4 | **0.989** | 0.902 | 0.940 |
| mean | 0.987 | 0.929 | 0.957 |

*Figure 6: The rows are parametrized by the parameter* $c_2$ *and columns by the parameter* $tr$ *of the Gripper reward function (see Section 4.1). The table presents results of 16 experiments.*

[3] In a separate experiments we have established that omitting the Gripper results in a very poor performance.

of the Reacher. We suspect that such a behavior will occur in a case of simple tasks, which can be acquired directly, as micro-actions are more expressive and can be tuned better. We compare this with the gripper+micro experiment, noticing that it achieves very good results and better stability with respect to the hyper-parameters. This might seem surprising at first, a tentative explanation is that more macro-actions leads to more parameters of the neural network leading to harder optimization problem. In the gripper+s-reacher experiment we address the problem of gluing by using a Reacher, similar to ReacherStabilized in Section 4.2.2. This again leads to much better performance compared to the base experiment.

| | gripper+micro | | | gripper+reacher | | | gripper+reacher+micro | | | gripper+s-reacher | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.03 | 0.06 | mean | 0.03 | 0.06 | mean | 0.03 | 0.06 | mean | 0.03 | 0.06 | mean |
| 3 | 0.86 | 0.74 | 0.80 | 0.17 | 0.08 | 0.13 | 0.92 | 0.01 | 0.46 | 0.88 | 0.69 | 0.81 |
| 4 | 0.80 | 0.65 | 0.72 | 0.32 | 0.15 | 0.23 | 0.75 | 0.01 | 0.40 | 0.87 | 0.70 | 0.79 |
| mean | 0.83 | 0.70 | 0.76 | 0.24 | 0.11 | 0.18 | 0.83 | 0.01 | 0.43 | 0.88 | 0.70 | 0.80 |
| max | | | 0.96 | | | 0.64 | | | 0.99 | | | 0.99 |

*Figure 7: The rows are parametrized by the parameter $c_2$ and columns by the parameter $tr$ of the Gripper reward function (see Section 4.1). The table presents results of approximately 100 experiments.*

### 4.3.2 Non-hierarchical experiments

**Ad. Experiments 1,2.** Figure 8 shows limited ability to generalize in the non-hierarchical case. Every dot represents a single training which lasted through 50 millions interactions with the environment. All hyperparameters are fixed except for the maximal angle $\alpha$ from which the initial position of the capsule and the initial position of the arm are drawn. After $k$-th episode we increase the angle to $k \cdot \alpha$. In total 50 millions interactions happen over $10^5$ episodes, hence the minimal $\alpha$ for which at the end of the experiment we reach the full circle is $\alpha = \frac{\pi}{10^5}$. More results for this specific hyperparameter can be found in Appendix B.2. On the horizontal axis are marked approximate values of $10^5 \cdot \alpha$ at the end of training (logarithmic scale). Graphs in Figure 8 are shown for $tr = 0.08$, $c_1 = -1$, $c_2 = 2.1$, $c_3 = -4$.

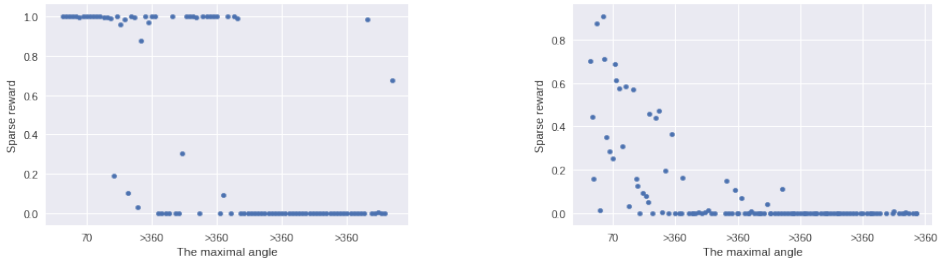


*Figure 8: The vertical axis shows sparse rewards in non-hierarchical experiments 1 (left) and 2 (right). The horizontal axis shows the max. angle reached after 50 millions steps. Definition 1 describes the sparse reward.*

## 5 Conclusion and further work

In this work we considered Hierarchical Reinforcement Learning with Parameters (HRLP) and tested this model on **reach-and-move** and **reach-and-grip** tasks. From experiments in Section 4 we conclude that HRLP behaves reasonably well in practice and from the mathematical analysis in Section 3 follows that combined with techniques proposed in [6, 1] it has good theoretical properties. Moreover, our approach seems to be conceptually and technically quite simple. This simplicity is paired with competitive experimental results and easy to analyze mathematical structure.

As a further work we are planning to test HRLP on a camera input and on a real robot. We are also planning to combine HRLP with other algorithms, in particular with those derived from [2, 25]. Another interesting direction would be building a library of re-usable polices for standard tasks and standard robotic models. Recently very interesting attempts have been made to algorithmically determine the hierarchical structure of a given problem [30, 22]. It would be interesting to extract re-usable macro-actions generated by those algorithms. More details about possible further HRLP experiments are presented in Appendix C.

**Acknowledgments**

This research would not be possible without a support from the PL-Grid Infrastructure. In particular, we used extensively the Prometheus supercomputer, located in the Academic Computer Center Cyfronet in the AGH University of Science and Technology in Kraków, Poland. We also express our gratitude to Goodeep for providing servers needed at the initial stage of the experiment.

We are very grateful for a number of insightful comments from the three anonymous referees. The comments helped to shape the final form of this paper and will be beneficial for our further research.

Last but not least, we would like to thank our intern Tomasz Gąsior, responsible for completely different tasks, but who nevertheless helped us on the text-processing front in the hectic days before the submission to CoRL.

## References

[1] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust Region Policy Optimization. In *ICML 2015*.

[2] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. URL http://arxiv.org/abs/1611.02247.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. 2015. URL http://arxiv.org/abs/1504.00702.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. 2015. URL http://arxiv.org/abs/1509.02971.

[5] I. Popov, N. Heess, T. P. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. A. Riedmiller. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. URL http://arxiv.org/abs/1704.03073.

[6] J. Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, EECS Department, University of California, Berkeley.

[7] R. S. Sutton, D. Precup, and S. P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[8] M. Wiering and J. Schmidhuber. Hq-learning. *Adaptive Behaviour*, 6(2):219–246, 1997.

[9] C. Florensa, Y. Duan, and P. Abbeel. Stochastic neural networks for hierarchical reinforcement learning. URL http://arxiv.org/abs/1704.03012.

[10] N. Heess, G. Wayne, Y. Tassa, T. P. Lillicrap, M. A. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. URL http://arxiv.org/abs/1610.05182.

[11] J. E. Pratt, C. Chew, A. Torres, P. Dilworth, and G. A. Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *I. J. Robotics Res.*, 20(2):129–143, 2001.

[12] E. Todorov, W. Li, and X. Pan. From task parameters to motor synergies: A hierarchical framework for approximately optimal control of redundant manipulators. *J. Field Robotics*, 22 (11):691–710, 2005.

[13] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *NIPS 2016*.

[14] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML 2002, Sydney, Australia, 2002*, pages 267–274, 2002.

[15] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ, IROS 2012, Vilamoura, Portugal, October 7-12, 2012*, pages 5026–5033, 2012.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. URL http://arxiv.org/abs/1606.01540.

[17] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. 2016. URL http://arxiv.org/abs/1604.06778.

[18] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. URL http://arxiv.org/abs/1611.01224.

[19] OpenAI. Requests for Research: Better sample efficiency for TRPO, 2017. URL https://openai.com/requests-for-research/#better-sample-efficiency-for-trpo.

[20] B. C. da Silva, G. Konidaris, and A. Barto. Active Learning of Parameterized Skills. In *ICML 2014*.

[21] C. Daniel, H. van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016.

[22] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. 2017. URL http://arxiv.org/abs/1703.01161.

[23] N. Mehta, S. Ray, P. Tadepalli, and T. G. Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *ICML 2008, Finland, 2008*.

[24] W. Masson and G. Konidaris. Reinforcement Learning with Parameterized Actions. *AAAI 2016*.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML 2016*.

[26] R. S. Sutton and A. G. Barto. *Reinforcement learning - an introduction*. MIT Press, 1998.

[27] I. Csiszár. $I$-divergence geometry of probability distributions and minimization problems. *Ann. Probability*, 3:146–158, 1975.

[28] N. Gozlan and C. Léonard. Transport Inequalities. A Survey. 2010. ISSN 1024-2953. URL http://arxiv.org/abs/1003.3852.

[29] OpenAI. Faulty reward functions in the wild, 2017. URL https://blog.openai.com/faulty-reward-functions/.

[30] D. Held, X. Geng, C. Florensa, and P. Abbeel. Automatic Goal Generation for Reinforcement Learning Agents. 2017. URL http://arxiv.org/abs/1705.06366.

[31] F. den Hollander. Probability theory: The coupling method. 2012. URL http://websites.math.leidenuniv.nl/probability/lecturenotes/CouplingLectures.pdf.

[32] J. Schulman. Modular RL, an implementation of TRPO, PPO and CEM. https://github.com/joschu/modular_rl, 2016.

[33] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. 2016. URL http://arxiv.org/abs/1605.02688.

[34] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS 2011*, pages 627–635.

[35] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *ICML 2000*.

[36] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences, 2017. URL https://arxiv.org/abs/1706.03741.

# A Proofs

## A.1 Proof of Theorem 1

*Proof.* (sketch) The proof in [1] is neatly based on a coupling argument which also works in the hierarchical case. According to [1, Def. 1] we say that a pair of policies is $\alpha$-coupled for some $\alpha \in (0,1)$ if there exists a $\mathcal{A}^2$-valued policy $\hat{\pi}$ such that marginal distributions of $\hat{\pi}((a,\tilde{a})|s)$ are $\pi(a|s)$, $\tilde{\pi}(\tilde{a}|s)$ and such that

$$\sup_{s\in\mathcal{S}} \mathbb{P}_{\hat{\pi}}(a \neq \tilde{a}|s) \leq \alpha. \tag{7}$$

Using [31, Theorem 2.12] one can find $\alpha$-coupling with $\alpha \leq D_{TV}^{\max}(\pi,\tilde{\pi})$. The rest of the argument follows the lines of [1, App. A]. □

## A.2 Proof of Lemma 1

*Proof.* We recall that

$$\mu = \sum_{k=1}^{n} p_k 1_{x\in\mathcal{A}_k}\mu_k, \quad \tilde{\mu} = \sum_{k=1}^{n} \tilde{p}_k 1_{x\in\mathcal{A}_k}\tilde{\mu}_k.$$

The Radon-Nikodym derivatives $\frac{\mathrm{d}\mu}{\mathrm{d}\tilde{\mu}}$ exist and we calculate

$$\frac{\mathrm{d}\mu}{\mathrm{d}\tilde{\mu}} = \sum_{k=1}^{n} \frac{p_k}{\tilde{p}_k} 1_{x\in\mathcal{S}_k} \frac{d\mu}{d\tilde{\mu}}.$$

Now we can calculate the Kullback-Leibler divergence

$$\begin{aligned} D_{KL}(\mu,\tilde{\mu}) &= \int_{\mathcal{A}} \log\left(\frac{\mathrm{d}\mu}{\mathrm{d}\tilde{\mu}}\right)\mathrm{d}\mu = \sum_{k=1}^{n} p_k \int_{\mathcal{A}_k} \log\left(\frac{\mathrm{d}\mu_k}{\mathrm{d}\tilde{\mu}_k}\right)\mathrm{d}\mu_k \\ &= \sum_{k=1}^{n} p_k \int_{\mathcal{A}_k} \left(\log\frac{p_k}{\tilde{p}_k} + \log\frac{\mathrm{d}\mu_k}{\mathrm{d}\tilde{\mu}_k}\right)\mathrm{d}\mu_k \\ &= \sum_{k=1}^{n} p_k \log\frac{p_k}{\tilde{p}_k} + \sum_{k=1}^{n} p_k \int_{\mathcal{A}_k} \left(\log\frac{\mathrm{d}\mu_k}{\mathrm{d}\tilde{\mu}_k}\right)\mathrm{d}\mu_k. \end{aligned}$$

The last expression is equal to the right-hand side of (1). □

## A.3 Proof of Lemma 2

*Proof.* Let $\hat{\pi}$ be the $\alpha$-couping of $\pi$ and $\tilde{\pi}$ described in the previous proof. We calculate

$$\begin{aligned} \left|\mathbb{E}_{\tilde{\pi}(\tilde{a}|s)}[A^{\pi}(s,\tilde{a})]\right| &= \left|\mathbb{E}_{\hat{\pi}((a,\tilde{a})|s)}[A^{\pi}(s,\tilde{a})]\right| \\ &\leq \left|\mathbb{E}_{\hat{\pi}((a,\tilde{a})|s)}[A^{\pi}(s,a)]\right| + \mathbb{E}_{\hat{\pi}((a,\tilde{a})|s)}\left|A^{\pi}(s,\tilde{a}) - A^{\pi}(s,a)\right|. \end{aligned}$$

The first summand is 0 by the definition of the advantage and the fact that $a$ under is distributed according to $\pi$. One can readily check that for any policy $\pi$ and $s \in \mathcal{S}$ and $a \in \mathcal{A}$ we have $|Q^{\pi}(s,a)| \leq R/(1-\gamma)$ and $V^{\pi}(s) \leq R/(1-\gamma)$. Thus $|A^{\pi}(s,a)| \leq 2R/(1-\gamma)$. Further we estimate

$$\sup_{s\in\mathcal{S}} \mathbb{E}_{\hat{\pi}((a,\tilde{a})|s)}\left|A^{\pi}(s,\tilde{a}) - A^{\pi}(s,a)\right| \leq \frac{4R}{1-\gamma}\mathbb{P}_{\hat{\pi}}(a \neq \tilde{a}) \leq \frac{4R}{1-\gamma}\alpha.$$

This concludes the proof of inequality (3). □

# B Additional information on experiments

## B.1 Experimental setting

Experiments were conducted using Modular RL framework [32] with Theano [33] library.We describe below the model behind the Manager and the model of the arm used for simulation in the MuJoCo [15] engine. In Section 4.1 we describe three non-hierarchical experiments in which we trained Grippers, Movers and Reachers. In Sections 4.2 and 4.3 we describe hierarchical experiments related to the **reach-and-move** and **reach-and-grip** task respectively.

**The model of the Manager.** The architecture of the neural network which we used to model the Manager consists of the input layer of the size of the state space, 2 dense layers and the output layer which is used to decide about the macro-action and its parameters. An example of a model of the Manager is presented in Figure 9

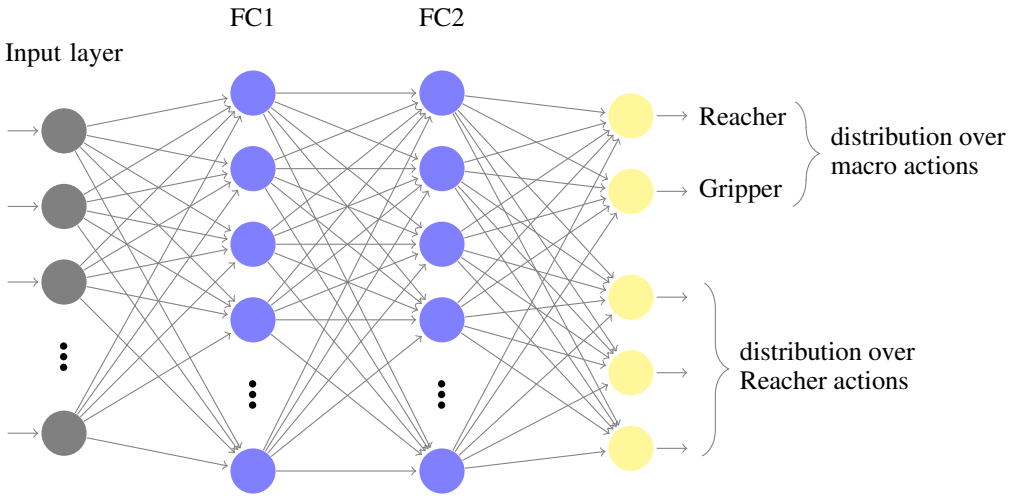


*Figure 9: A model of the Manager for* **reach-and-grip** *task without micro-actions. Notice that the output contains* 3 *parameters which are passed to the Reacher.*

**Mujoco models.** In our experiments we used the `Arm Claw` robotic arm provided in the MuJoCo package [15], see Figure 3. The model has 9 joints and the state space consists in total of 24 dimensions, including 9 joints' angles, 9 joints' velocities, 3 dimensional position of the claw and 3 dimensional position of the target.

## B.2 Dependence on the KL divergence

In the context of experiments described in Section 4.3.2, Figure 8, we verified obtaining of KL divergence for an experiment which guarantees that at its end the full range of the capsule and the arm reaches 360 degrees, that is in principle the agent should be able to reach the capsule for an arbitrary considered entrance position.

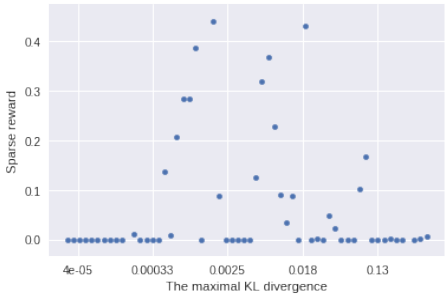


*Figure 10: Figure shows successes in obtaining alternative rewards in the non-hierarchical experiment 2. The horizontal axis (logarithmic scale) shows the maximal KL divergence between the new and old policy. For the definition of the sparse reward see Definition 1.*

## C Detailed further experiments with HRLP

- In the process of training the Manager collect points from which more difficult non-hierarchical tasks such as Mover, should be retrained. Use this idea combined with the DAGGER algorithm [34].
- Verify dependency of hierarchical experiments on the maximal KL divergence, the length of an action executed by the Manager, and the size of the Manager network.
- Verify implications of imposing a bound on micro-actions in terms of time or maximal force; verify implications of introducing penalties for using micro-actions.
- Discover better cost function for gripping and moving using reverse reinforcement learning [35] or a human-generated feedback [36].
- Combine two complicated tasks – in the current work we combined one more difficult task, such as moving, with an easier task, such as reaching.

## D Hyperparameters

Unless otherwise indicated, in our experiments we used the following hyperparameters for Trust Region Policy Optimization: maximal KL divergence between new and old policy $0.01$, $\gamma = 0.99$, the batch size $50000$ for the non-hierarchical experiments and $5000$ for hierarchical experiments, the path length 300-500 for a single episode.

The policy network (see Figure 9) in all tasks consists of two hidden layers with $64$ units each.