

# Algorithms for Multi-task Reinforcement Learning

*Alexander Li*  
*Pieter Abbeel, Ed.*

Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/Eecs-2020-110

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/Eecs-2020-110.html>

May 29, 2020



Copyright © 2020, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Algorithms for Multi-task Reinforcement Learning

by

Alexander C. Li

A thesis submitted in partial satisfaction of the  
requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

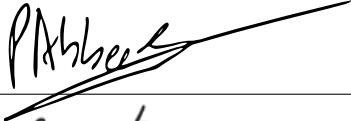

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair  
Professor Sergey Levine

Spring 2020

The thesis of Alexander C. Li, titled Algorithms for Multi-task Reinforcement Learning, is approved:

Chair		Date	<u>29 - MAY - 2020</u>
		Date	<u>May 29, 2020</u>

University of California, Berkeley

# Algorithms for Multi-task Reinforcement Learning

Copyright 2020  
by  
Alexander C. Li

## Abstract

## Algorithms for Multi-task Reinforcement Learning

by

Alexander C. Li

Master of Science in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

Machine learning has made great strides on a variety of tasks, including image classification, natural language understanding, robotic control, and game-playing. However, much less progress has been made on efficient multi-task learning: generalist algorithms that can leverage experience on previous tasks to quickly excel at new ones. Such algorithms are necessary in order to efficiently perform new tasks when data, compute, time, or energy is limited. In this thesis, we develop two novel algorithms for multi-task reinforcement learning.

First, we examine the potential for improving cross-task generalization in hierarchical reinforcement learning. We derive a new hierarchical policy gradient with an unbiased latent-dependent baseline, and we introduce Hierarchical Proximal Policy Optimization (HiPPO), an on-policy method to efficiently train all levels of the hierarchy jointly. This allows us to discover robust and transferable skills, and quickly learn how to perform a new task by finetuning skills learned on similar environments.

Second, we introduce Generalized Hindsight, which is based on the insight that unsuccessful attempts to solve one task are often a rich source of information for other tasks. Generalized Hindsight is an approximate inverse reinforcement learning technique that matches generated behaviors with the tasks they are best suited for, before being used by an off-policy RL optimizer. Generalized Hindsight is substantially more sample-efficient than standard relabeling techniques, which we empirically demonstrate on a suite of multi-task navigation and manipulation tasks.

To my parents, William Li and Elizabeth Xu.

# Contents

<b>Contents</b>	<b>ii</b>
<b>List of Figures</b>	<b>iii</b>
<b>List of Tables</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Hierarchical Policies for Multi-task Learning . . . . .	1
1.3 Sharing Data Across Tasks . . . . .	2
<b>2 Sub-policy Adaptation for Hierarchical Reinforcement Learning</b>	<b>3</b>
2.1 Acknowledgements . . . . .	3
2.2 Introduction . . . . .	3
2.3 Preliminaries . . . . .	4
2.4 Related Work . . . . .	5
2.5 Efficient Hierarchical Policy Gradients . . . . .	6
2.6 Experiments . . . . .	12
2.7 Conclusions and Future Work . . . . .	17
<b>3 Generalized Hindsight for Reinforcement Learning</b>	<b>18</b>
3.1 Acknowledgements . . . . .	18
3.2 Introduction . . . . .	18
3.3 Background . . . . .	20
3.4 Generalized Hindsight . . . . .	22
3.5 Experimental Evaluation . . . . .	25
3.6 Related Work . . . . .	30
3.7 Conclusions . . . . .	31
<b>4 Conclusion</b>	<b>32</b>
<b>Bibliography</b>	<b>33</b>



# List of Figures

2.1	Temporal hierarchy studied in this paper. A latent code $z_t$ is sampled from the manager policy $\pi_{\theta_h}(z_t s_t)$ every $p$ time-steps, using the current observation $s_{kp}$ . The actions $a_t$ are sampled from the sub-policy $\pi_{\theta_l}(a_t s_t, z_{kp})$ conditioned on the same latent code from $t = kp$ to $(k + 1)p - 1$ . . . . .	4
2.2	Environments used to evaluate the performance of our method. Every episode has a different configuration: wall heights for (a)-(b) and ball positions for (c)-(d)	12
2.3	Analysis of different time-commitment strategies on learning from scratch. . . . .	13
2.4	Using a skill-conditioned baseline, as defined in Section 2.5, generally improves performance of HiPPO when learning from scratch. . . . .	14
2.5	Comparison of HiPPO and HierVPG to prior hierarchical methods on learning from scratch. . . . .	14
2.6	Benefit of adapting some given skills when the preferences of the environment are different from those of the environment where the skills were originally trained. Adapting skills with HiPPO has better learning performance than leaving the skills fixed or learning from scratch. . . . .	16
2.7	HIRO performance on Ant Gather with and without access to the ground truth $(x, y)$ , which it needs to communicate useful goals. . . . .	17
3.1	A rollout can often provide very little information about how to perform a task A. In the trajectory-following task on the left, the trajectory (green) sees almost no reward signal (areas in red). However, in the multi-task setting where each target trajectory represents a different task, we can find another task B for which our trajectory is a “pseudo-demonstration.” This hindsight relabeling provides high reward signal and enables sample-efficient learning. . . . .	19
3.2	Trajectories $\tau(z_i)$ , collected trying to maximize $r(\cdot z_i)$ , may contain very little reward signal about how to solve their original tasks. Generalized Hindsight checks against randomly sampled “candidate tasks” $\{v_i\}_{i=1}^K$ to find different tasks $z'_i$ for which these trajectories are “pseudo-demonstrations.” Using off-policy RL, we can obtain more reward signal from these relabeled trajectories. . . . .	20

3.3	Environments we report comparisons on. PointTrajectory requires a 2D pointmass to follow a target trajectory; PointReacher requires moving the pointmass to a goal location, while avoiding an obstacle and modulating its energy usage. In (b), the red circle indicates the goal location, while the blue triangle indicates an imagined obstacle to avoid. Fetch has the same reward formulation as PointReacher, but requires controlling the noisy Fetch robot in 3 dimensions. HalfCheetah requires learning running in both directions, flipping, jumping, and moving efficiently. AntDirection requires moving in a target direction as fast as possible. . . . .	25
3.4	Learning curves comparing Generalized Hindsight algorithms to baseline methods. For environments with a goal-reaching component, we also compare to HER. In (a), AIR learning curve obscures the Advantage learning curve. In (d) and (e), where we use $N = 500$ for AIR, AIR takes much longer to run than the other methods. 10 seeds were used for all runs. . . . .	27
3.5	The agents efficiently learn a wide range of behaviors. On HalfCheetahMultiObjective, the robot can stay still to conserve energy, run quickly forwards and backwards, and do a frontflip. On AntDirection, the robot can run quickly in any given direction. . . . .	28
3.6	Red denotes areas of high reward, for following a target trajectory (top) or reaching a goal (bottom). Blue indicates areas of negative reward, where an obstacle may be placed. On both environments, relabeling finds tasks on which our trajectory has high reward signal. On PointReacher, AIR does not place the obstacle arbitrarily far. It places the relabeled obstacle within the curve of the trajectory, since this is the only way that the curved path would be better than a straight-line path (that would come close to the relabeled obstacle). . . . .	29
3.7	Comparison of relabeling fidelity on optimal trajectories for approximate IRL, advantage relabeling, and reward relabeling. We train a multi-task policy to convergence on the PointReacher environment. We roll out our policy on 1000 randomly sampled tasks $z$ , and apply each relabeling method to select from $K = 100$ randomly sampled tasks $v$ . For approximate IRL, we compare against $N = 10$ prior trajectories. The x-axis shows the weight on energy for the task $z$ used for the rollout, while the y-axis shows the weight on energy for the relabeled task $z'$ . Note that goal location, obstacle location, and weights on their rewards/penalties are varying as well, but are not shown. Closer to to the line $y = x$ indicates higher fidelity, since it implies $z' \approx z^*$ . . . . .	30

# List of Tables

2.1	Zero-shot transfer performance. The final return in the initial environment is shown, as well as the average return over 25 rollouts in each new modified environment.	15
2.2	Empirical evaluation of Lemma 1. In the middle and right columns, we evaluate the quality of our assumption by computing the largest probability of a certain action under other skills ( $\epsilon$ ), and the action probability under the actual latent. We also report the cosine similarity between our approximate gradient and the exact gradient from Eq. 2.3. The mean and standard deviation of these values are computed over the full batch collected at iteration 10.	16

## Acknowledgments

I would like to thank my advisor, Professor Pieter Abbeel. From the beginning, he has inspired me to dream big, while believing in me and guiding me the entire way through. I'm incredibly grateful to my mentors, Carlos Florensa and Lerrel Pinto. From them, I've learned how to ask the right questions, design the right experiments, and become a better researcher. I'll also always fondly remember the times I spent with my brilliant peers on the 4th floor of Sutardja Dai Hall – from late night research discussions to lazy summer frisbee games on the Glade. Most importantly, I thank my family. Their unwavering love and support has made everything possible.

# Chapter 1

## Introduction

### 1.1 Motivation

Model-free reinforcement learning (RL) combined with powerful function approximators has achieved remarkable success in games like Atari [54] and Go [84], and control tasks like walking [27] and flying [39]. However, a key limitation to these methods is their sample complexity. They often require millions of samples to learn simple locomotion skills, and sometimes even billions of samples to learn more complex game strategies. Creating general purpose agents will necessitate learning multiple such skills or strategies, which further exacerbates the inefficiency of these algorithms. On the other hand, humans (or biological agents) are not only able to learn a multitude of different skills, but from orders of magnitude fewer samples [38]. So, how do we endow RL agents with this ability to learn efficiently across multiple tasks?

### 1.2 Hierarchical Policies for Multi-task Learning

Hierarchical RL is one approach towards distilling knowledge across tasks. Hierarchical methods decompose a task into a sequence of smaller goals to be achieved. A higher-level reasoning module, which can just be a deep neural network, chooses these goals in an online manner, while a lower-level module handles the mechanics of actually carrying out each instruction. Previous works frequently solely focus on how hierarchical RL can accelerate learning on a single task. In Chapter 2, we explore how exactly hierarchical RL can improve multi-task learning. We develop a novel algorithm called HiPPO that learns skills end-to-end; we show that these skills are easily transferable and robust to a variety of changes to the reward functions and dynamics. We also show that finetuning skills with HiPPO substantially improves learning efficiency and final performance on a target task.

### 1.3 Sharing Data Across Tasks

Multi-task learning ability can also improve by directly sharing data across tasks. After all, a policy simply distills the data into a functional representation. In standard multi-task RL settings, low-reward data collected while trying to solve one task provides little to no signal for solving that particular task and is hence effectively wasted. In Chapter 3, we argue that this data, which is uninformative for one task, is likely a rich source of information for other tasks. To leverage this insight and efficiently reuse data, we present Generalized Hindsight: an approximate inverse reinforcement learning technique for relabeling behaviors with the right tasks. Intuitively, given a behavior generated under one task, Generalized Hindsight returns a different task that the behavior is better suited for. Then, the behavior is relabeled with this new task before being used by an off-policy RL optimizer. Compared to standard relabeling techniques, Generalized Hindsight is up to  $5\times$  more sample-efficient, which we empirically demonstrate on a suite of multi-task navigation and manipulation tasks.

# Chapter 2

## Sub-policy Adaptation for Hierarchical Reinforcement Learning

### 2.1 Acknowledgements

The work in this section was conducted, written up, and published at the *International Conference on Learning Representations (ICLR) 2020* by the author, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel [48]. Both the author and Carlos Florensa contributed equally.

### 2.2 Introduction

Reinforcement learning (RL) has made great progress in a variety of domains, from playing games such as Pong and Go [54, 86] to automating robotic locomotion [80, 30], dexterous manipulation [20, 61], and perception [58, 21]. Yet, most work in RL is still learning from scratch when faced with a new problem. This is particularly inefficient when tackling multiple related tasks that are hard to solve due to sparse rewards or long horizons.

A promising technique to overcome this limitation is hierarchical reinforcement learning (HRL) [90]. In this paradigm, policies have several modules of abstraction, allowing to reuse subsets of the modules. The most common case consists of temporal hierarchies [70, 13], where a higher-level policy (manager) takes actions at a lower frequency, and its actions condition the behavior of some lower level skills or sub-policies. When transferring knowledge to a new task, most prior works fix the skills and train a new manager on top. Despite having a clear benefit in kick-starting the learning in the new task, having fixed skills can considerably cap the final performance on the new task [19]. Little work has been done on adapting pre-trained sub-policies to be optimal for a new task.

In this work, we develop a new framework for simultaneously adapting all levels of temporal hierarchies. First, we derive an efficient approximated hierarchical policy gradient. The key insight is that, despite the decisions of the manager being unobserved latent variables from the point of view of the Markovian environment, from the perspective of the sub-policies

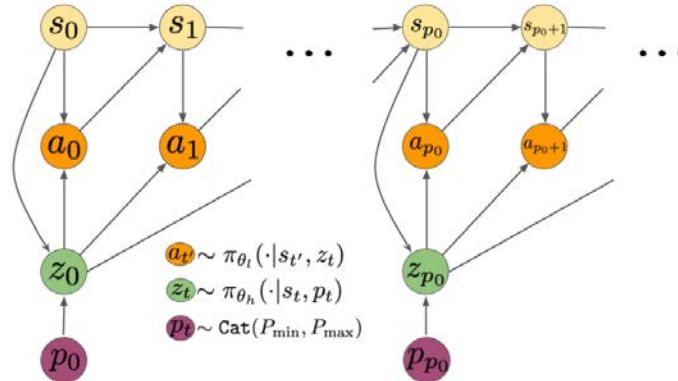


Figure 2.1: Temporal hierarchy studied in this paper. A latent code  $z_t$  is sampled from the manager policy  $\pi_{\theta_h}(z_t|s_t)$  every  $p$  time-steps, using the current observation  $s_{kp}$ . The actions  $a_t$  are sampled from the sub-policy  $\pi_{\theta_l}(a_t|s_t, z_{kp})$  conditioned on the same latent code from  $t = kp$  to  $(k + 1)p - 1$

they can be considered as part of the observation. We show that this provides a decoupling of the manager and sub-policy gradients, which greatly simplifies the computation in a principled way. It also theoretically justifies a technique used in other prior works [22]. Second, we introduce a sub-policy specific baseline for our hierarchical policy gradient. We prove that this baseline is unbiased, and our experiments reveal faster convergence, suggesting efficient gradient variance reduction. Then, we introduce a more stable way of using this gradient, Hierarchical Proximal Policy Optimization (HiPPO). This method helps us take more conservative steps in our policy space [79], critical in hierarchies because of the interdependence of each layer. Results show that HiPPO is highly efficient both when learning from scratch, i.e. adapting randomly initialized skills, and when adapting pretrained skills on a new task. Finally, we evaluate the benefit of randomizing the time-commitment of the sub-policies, and show it helps both in terms of final performance and zero-shot adaptation on similar tasks.

### 2.3 Preliminaries

We define a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple  $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, H)$ , where  $\mathcal{S}$  is a state set,  $\mathcal{A}$  is an action set,  $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}_+$  is the transition probability distribution,  $\gamma \in [0, 1]$  is a discount factor, and  $H$  the horizon. Our objective is to find a stochastic policy  $\pi_\theta$  that maximizes the expected discounted return within the MDP,  $\eta(\pi_\theta) = \mathbb{E}_\tau[\sum_{t=0}^H \gamma^t r(s_t, a_t)]$ . We use  $\tau = (s_0, a_0, \dots)$  to denote the entire state-action trajectory, where  $s_0 \sim \rho_0(s_0)$ ,  $a_t \sim \pi_\theta(a_t|s_t)$ , and  $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$ .

In this work, we propose a method to learn a hierarchical policy and efficiently adapt all the levels in the hierarchy to perform a new task. We study hierarchical policies composed



of a higher level, or manager  $\pi_{\theta_h}(z_t|s_t)$ , and a lower level, or sub-policy  $\pi_{\theta_l}(a_{t'}|z_t, s_{t'})$ . The higher level does not take actions in the environment directly, but rather outputs a command, or latent variable  $z_t \in \mathcal{Z}$ , that conditions the behavior of the lower level. We focus on the common case where  $\mathcal{Z} = \mathbb{Z}_n$  making the manager choose among  $n$  sub-policies, or skills, to execute. The manager typically operates at a lower frequency than the sub-policies, only observing the environment every  $p$  time-steps. When the manager receives a new observation, it decides which low level policy to commit to for  $p$  environment steps by the means of a latent code  $z$ . Figure 2.1 depicts this framework where the high level frequency  $p$  is a random variable, which is one of the contribution of this paper as described in Section 2.5. Note that the class of hierarchical policies we work with is more restrictive than others like the options framework, where the time-commitment is also decided by the policy. Nevertheless, we show that this loss in policy expressivity acts as a regularizer and does not prevent our algorithm from surpassing other state-of-the art methods.

## 2.4 Related Work

There has been growing interest in HRL for the past few decades [90, 70], but only recently has it been applied to high-dimensional continuous domains as we do in this work [42, 12]. To obtain the lower level policies, or skills, most methods exploit some additional assumptions, like access to demonstrations [43, 52, 72, 82], policy sketches [2], or task decomposition into sub-tasks [23, 88]. Other methods use a different reward for the lower level, often constraining it to be a “goal reacher” policy, where the signal from the higher level is the goal to reach [56, 45, 98]. These methods are very promising for state-reaching tasks, but might require access to goal-reaching reward systems not defined in the original MDP, and are more limited when training on tasks beyond state-reaching. Our method does not require any additional supervision, and the obtained skills are not constrained to be goal-reaching.

When transferring skills to a new environment, most HRL methods keep them fixed and simply train a new higher-level on top [29, 31]. Other work allows for building on previous skills by constantly supplementing the set of skills with new ones [83], but they require a hand-defined curriculum of tasks, and the previous skills are never fine-tuned. Our algorithm allows for seamless adaptation of the skills, showing no trade-off between leveraging the power of the hierarchy and the final performance in a new task. Other methods use invertible functions as skills [26], and therefore a fixed skill can be fully overwritten when a new layer of hierarchy is added on top. This kind of “fine-tuning” is promising, although similar to other works [64], they do not apply it to temporally extended skills as we do here.

One of the most general frameworks to define temporally extended hierarchies is the options framework [90], and it has recently been applied to continuous state spaces [5]. One of the most delicate parts of this formulation is the termination policy, and it requires several regularizers to avoid skill collapse [28, 99]. This modification of the objective may be difficult to tune and affects the final performance. Instead of adding such penalties, we propose to have skills of a random length, not controlled by the agent during training of

the skills. The benefit is two-fold: no termination policy to train, and more stable skills that transfer better. Furthermore, these works only used discrete action MDPs. We lift this assumption, and show good performance of our algorithm in complex locomotion tasks. There are other algorithms recently proposed that go in the same direction, but we found them more complex, less principled (their per-action marginalization cannot capture well the temporal correlation within each option), and without available code or evidence of outperforming non-hierarchical methods [87].

The closest work to ours in terms of final algorithm structure is the one proposed by Frans et al. [22]. Their method can be included in our framework, and hence benefits from our new theoretical insights. We introduce a modification that is shown to be highly beneficial: the random time-commitment mentioned above, and find that our method can learn in difficult environments without their complicated training scheme.

## 2.5 Efficient Hierarchical Policy Gradients

When using a hierarchical policy, the intermediate decision taken by the higher level is not directly applied in the environment. Therefore, technically it should not be incorporated into the trajectory description as an observed variable, like the actions. This makes the policy gradient considerably harder to compute. In this section we first prove that, under mild assumptions, the hierarchical policy gradient can be accurately approximated without needing to marginalize over this latent variable. Then, we derive an unbiased baseline for the policy gradient that can reduce the variance of its estimate. Finally, with these findings, we present our method, Hierarchical Proximal Policy Optimization (HiPPO), an on-policy algorithm for hierarchical policies, allowing learning at all levels of the policy jointly and preventing sub-policy collapse.

### Approximate Hierarchical Policy Gradient

Policy gradient algorithms are based on the likelihood ratio trick [101] to estimate the gradient of returns with respect to the policy parameters as

$$\nabla_{\theta} \eta(\pi_{\theta}) = \mathbb{E}_{\tau} [\nabla_{\theta} \log P(\tau) R(\tau)] \approx \frac{1}{N} \sum_{i=1}^n \nabla_{\theta} \log P(\tau_i) R(\tau_i) \quad (2.1)$$

$$= \frac{1}{N} \sum_{i=1}^n \frac{1}{H} \sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau_i) \quad (2.2)$$

In a temporal hierarchy, a hierarchical policy with a manager  $\pi_{\theta_h}(z_t | s_t)$  selects every  $p$  time-steps one of  $n$  sub-policies to execute. These sub-policies, indexed by  $z \in \mathbb{Z}_n$ , can be represented as a single conditional probability distribution over actions  $\pi_{\theta_l}(a_t | z_t, s_t)$ . This allows us to not only use a given set of sub-policies, but also leverage skills learned with

Stochastic Neural Networks (SNNs) [19]. Under this framework, the probability of a trajectory  $\tau = (s_0, a_0, s_1, \dots, s_H)$  can be written as

$$P(\tau) = \left( \prod_{k=0}^{H/p} \left[ \sum_{j=1}^n \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j) \right] \right) \left[ P(s_0) \prod_{t=1}^H P(s_{t+1} | s_t, a_t) \right]. \quad (2.3)$$

The mixture action distribution, which presents itself as an additional summation over skills, prevents additive factorization when taking the logarithm, as from Eq. 2.1 to 2.2. This can yield numerical instabilities due to the product of the  $p$  sub-policy probabilities. For instance, in the case where all the skills are distinguishable all the sub-policies' probabilities but one will have small values, resulting in an exponentially small value. In the following Lemma, we derive an approximation of the policy gradient, whose error tends to zero as the skills become more diverse, and draw insights on the interplay of the manager actions.

**Lemma 1.** *If the skills are sufficiently differentiated, then the latent variable can be treated as part of the observation to compute the gradient of the trajectory probability. Let  $\pi_{\theta_h}(z|s)$  and  $\pi_{\theta_l}(a|s, z)$  be Lipschitz functions w.r.t. their parameters, and assume that  $0 < \pi_{\theta_l}(a|s, z_j) < \epsilon \forall j \neq kp$ , then*

$$\nabla_{\theta} \log P(\tau) = \sum_{k=0}^{H/p} \nabla_{\theta} \log \pi_{\theta_h}(z_{kp} | s_{kp}) + \sum_{t=0}^H \nabla_{\theta} \log \pi_{\theta_l}(a_t | s_t, z_{kp}) + \mathcal{O}(nH\epsilon^{p-1}) \quad (2.4)$$

Our assumption can be seen as having diverse skills. Namely, for each action there is just one sub-policy that gives it high probability. In this case, the latent variable can be treated as part of the observation to compute the gradient of the trajectory probability. Many algorithms to extract lower-level skills are based on promoting diversity among the skills [19, 17], therefore usually satisfying our assumption. We further analyze how well this assumption holds in our experiments section and Table 2.2.

*Proof.* From the point of view of the MDP, a trajectory is a sequence  $\tau = (s_0, a_0, s_1, a_1, \dots, a_{H-1}, s_H)$ . Let's assume we use the hierarchical policy introduced above, with a higher-level policy modeled as a parameterized discrete distribution with  $n$  possible outcomes  $\pi_{\theta_h}(z|s) = \text{Categorical}_{\theta_h}(n)$ . We can expand  $P(\tau)$  into the product of policy and environment dynamics terms, with  $z_j$  denoting the  $j$ th possible value out of the  $n$  choices,

$$P(\tau) = \left( \prod_{k=0}^{H/p} \left[ \sum_{j=1}^n \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j) \right] \right) \left[ P(s_0) \prod_{t=1}^H P(s_{t+1} | s_t, a_t) \right]$$

Taking the gradient of  $\log P(\tau)$  with respect to the policy parameters  $\theta = [\theta_h, \theta_l]$ , the dynamics terms disappear, leaving:

$$\begin{aligned} \nabla_{\theta} \log P(\tau) &= \sum_{k=0}^{H/p} \nabla_{\theta} \log \left( \sum_{j=1}^n \pi_{\theta_l}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{s,\theta}(a_t | s_t, z_j) \right) \\ &= \sum_{k=0}^{H/p} \frac{1}{\sum_{j=1}^n \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j)} \sum_{j=1}^n \nabla_{\theta} \left( \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j) \right) \end{aligned}$$

The sum over possible values of  $z$  prevents the logarithm from splitting the product over the  $p$ -step sub-trajectories. This term is problematic, as this product quickly approaches 0 as  $p$  increases, and suffers from considerable numerical instabilities. Instead, we want to approximate this sum of products by a single one of the terms, which can then be decomposed into a sum of logs. For this we study each of the terms in the sum: the gradient of a sub-trajectory probability under a specific latent  $\nabla_{\theta} \left( \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j) \right)$ . Now we can use the assumption that the skills are easy to distinguish,  $0 < \pi_{\theta_l}(a_t | s_t, z_j) < \epsilon \forall j \neq kp$ . Therefore, the probability of the sub-trajectory under a latent different than the one that was originally sampled  $z_j \neq z_{kp}$ , is upper bounded by  $\epsilon^p$ . Taking the gradient, applying the product rule, and the Lipschitz continuity of the policies, we obtain that for all  $z_j \neq z_{kp}$ ,

$$\begin{aligned} \nabla_{\theta} \left( \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j) \right) &= \nabla_{\theta} \pi_{\theta_h}(z_j | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_j) + \\ &\quad \sum_{t=kp}^{(k+1)p-1} \pi_{\theta_h}(z_j | s_{kp}) (\nabla_{\theta} \pi_{\theta_l}(a_t | s_t, z_j)) \prod_{\substack{t'=kp \\ t' \neq t}}^{(k+1)p-1} \pi_{\theta_l}(a_{t'} | s_{t'}, z_j) \\ &= \mathcal{O}(p\epsilon^{p-1}) \end{aligned}$$

Thus, we can replace the summation over latents by the single term corresponding to the latent that was sampled at that time.

$$\begin{aligned} \nabla_{\theta} \log P(\tau) &= \sum_{k=0}^{H/p} \frac{1}{\pi_{\theta_h}(z_{kp} | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_{kp})} \nabla_{\theta} \left( P(z_{kp} | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_{kp}) \right) + \frac{nH}{p} \mathcal{O}(p\epsilon^{p-1}) \\ &= \sum_{k=0}^{H/p} \nabla_{\theta} \log \left( \pi_{\theta_h}(z_{kp} | s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t | s_t, z_{kp}) \right) + \mathcal{O}(nH\epsilon^{p-1}) \\ &= \mathbb{E}_{\tau} \left[ \left( \sum_{k=0}^{H/p} \nabla_{\theta} \log \pi_{\theta_h}(z_{kp} | s_{kp}) + \sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta_l}(a_t | s_t, z_{kp}) \right) \right] + \mathcal{O}(nH\epsilon^{p-1}) \end{aligned}$$

□

Interestingly, this is exactly  $\nabla_{\theta} P(s_0, z_0, a_0, s_1, \dots)$ . In other words, it's the gradient of the probability of that trajectory, where the trajectory now includes the variables  $z$  as if they were observed.

## Unbiased Sub-Policy Baseline

The policy gradient estimate obtained when applying the log-likelihood ratio trick as derived above is known to have large variance. A very common approach to mitigate this issue without biasing the estimate is to subtract a baseline from the returns [65]. It is well known that such baselines can be made state-dependent without incurring any bias. However, it is still unclear how to formulate a baseline for all the levels in a hierarchical policy, since an action dependent baseline does introduce bias in the gradient [97]. It has been recently proposed to use latent-conditioned baselines [100]. Here we go further and prove that, under the assumptions of Lemma 1, we can formulate an unbiased latent dependent baseline for the approximate gradient (Eq. 2.4).

**Lemma 2.** *For any functions  $b_h : \mathcal{S} \rightarrow \mathbb{R}$  and  $b_l : \mathcal{S} \times \mathcal{Z} \rightarrow \mathbb{R}$  we have:*

$$\mathbb{E}_{\tau} \left[ \sum_{k=0}^{H/p} \nabla_{\theta} \log P(z_{kp} | s_{kp}) b_h(s_{kp}) \right] = 0 \quad \text{and} \quad \mathbb{E}_{\tau} \left[ \sum_{t=0}^H \nabla_{\theta} \log \pi_{\theta_t}(a_t | s_t, z_{kp}) b_l(s_t, z_{kp}) \right] = 0$$

*Proof.* We can use the tower property as well as the fact that the interior expression only depends on  $s_{kp}$  and  $z_{kp}$ :

$$\begin{aligned} \mathbb{E}_{\tau} \left[ \sum_{k=0}^{H/p} \nabla_{\theta} \log P(z_{kp} | s_{kp}) b(s_{kp}) \right] &= \sum_{k=0}^{H/p} \mathbb{E}_{s_{kp}, z_{kp}} \left[ \mathbb{E}_{\tau \setminus s_{kp}, z_{kp}} \left[ \nabla_{\theta} \log P(z_{kp} | s_{kp}) b(s_{kp}) \right] \right] \\ &= \sum_{k=0}^{H/p} \mathbb{E}_{s_{kp}, z_{kp}} \left[ \nabla_{\theta} \log P(z_{kp} | s_{kp}) b(s_{kp}) \right] \end{aligned}$$

Then, we can write out the definition of the expectation and undo the gradient-log trick to prove that the baseline is unbiased.

$$\begin{aligned} \mathbb{E}_{\tau} \left[ \sum_{k=0}^{H/p} \nabla_{\theta} \log \pi_{\theta_h}(z_{kp} | s_{kp}) b(s_{kp}) \right] &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp}) b(s_{kp}) \int_{z_{kp}} \pi_{\theta_h}(z_{kp} | s_{kp}) \nabla_{\theta} \log \pi_{\theta_h}(z_{kp} | s_{kp}) dz_{kp} ds_{kp} \\ &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp}) b(s_{kp}) \nabla_{\theta} \int_{z_{kp}} \pi_{\theta_h}(z_{kp} | s_{kp}) dz_{kp} ds_{kp} \\ &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp}) b(s_{kp}) \nabla_{\theta} 1 ds_{kp} \\ &= 0 \end{aligned}$$

□

*Proof.* We'll follow the same strategy to prove the second equality: apply the tower property, express the expectation as an integral, and undo the gradient-log trick.

$$\begin{aligned}
 & \mathbb{E}_\tau \left[ \sum_{t=0}^H \nabla_\theta \log \pi_{\theta_t}(a_t | s_t, z_{kp}) b(s_t, z_{kp}) \right] \\
 &= \sum_{t=0}^H \mathbb{E}_{s_t, a_t, z_{kp}} \left[ \mathbb{E}_{\tau \setminus s_t, a_t, z_{kp}} \left[ \nabla_\theta \log \pi_{\theta_t}(a_t | s_t, z_{kp}) b(s_t, z_{kp}) \right] \right] \\
 &= \sum_{t=0}^H \mathbb{E}_{s_t, a_t, z_{kp}} \left[ \nabla_\theta \log \pi_{\theta_t}(a_t | s_t, z_{kp}) b(s_{kp}, z_{kp}) \right] \\
 &= \sum_{t=0}^H \int_{(s_t, z_{kp})} P(s_t, z_{kp}) b(s_t, z_{kp}) \int_{a_t} \pi_{\theta_t}(a_t | s_t, z_{kp}) \nabla_\theta \log \pi_{\theta_t}(a_t | s_t, z_{kp}) da_t dz_{kp} ds_t \\
 &= \sum_{t=0}^H \int_{(s_t, z_{kp})} P(s_t, z_{kp}) b(s_t, z_{kp}) \nabla_\theta 1 dz_{kp} ds_t \\
 &= 0
 \end{aligned}$$

□

Now we apply Lemma 1 and Lemma 2 to Eq. 2.1. By using the corresponding value functions as the function baseline, the return can be replaced by the Advantage function  $A(s_{kp}, z_{kp})$  (see details in Schulman et al. 78), and we obtain the following approximate policy gradient expression:

$$\hat{g} = \mathbb{E}_\tau \left[ \left( \sum_{k=0}^{H/p} \nabla_\theta \log \pi_{\theta_h}(z_{kp} | s_{kp}) A(s_{kp}, z_{kp}) \right) + \left( \sum_{t=0}^H \nabla_\theta \log \pi_{\theta_t}(a_t | s_t, z_{kp}) A(s_t, a_t, z_{kp}) \right) \right]$$

This hierarchical policy gradient estimate can have lower variance than without baselines, but using it for policy optimization through stochastic gradient descent still yields an unstable algorithm. In the next section, we further improve the stability and sample efficiency of the policy optimization by incorporating techniques from Proximal Policy Optimization 79.

## Hierarchical Proximal Policy Optimization

Using an appropriate step size in policy space is critical for stable policy learning. Modifying the policy parameters in some directions may have a minimal impact on the distribution over actions, whereas small changes in other directions might change its behavior drastically and hurt training efficiency 36. Trust region policy optimization (TRPO) uses a constraint on

---

**Algorithm 1** HiPPO Rollout

---

- 1: **Input:** skills  $\pi_{\theta_l}(a|s, z)$ , manager  $\pi_{\theta_h}(z|s)$ , time-commitment bounds  $P_{\min}$  and  $P_{\max}$ , horizon  $H$
- 2: Reset environment:  $s_0 \sim \rho_0$ ,  $t = 0$ .
- 3: **while**  $t < H$  **do**
- 4:   Sample time-commitment  $p \sim \text{Cat}([P_{\min}, P_{\max}])$
- 5:   Sample skill  $z_t \sim \pi_{\theta_h}(\cdot|s_t)$
- 6:   **for**  $t' = t \dots (t + p)$  **do**
- 7:     Sample action  $a_{t'} \sim \pi_{\theta_l}(\cdot|s_{t'}, z_t)$
- 8:     Observe new state  $s_{t'+1}$  and reward  $r_{t'}$
- 9:   **end for**
- 10:    $t \leftarrow t + p$
- 11: **end while**
- 12: **Output:**  $(s_0, z_0, a_0, s_1, a_1, \dots, s_H, z_H, a_H, s_{H+1})$

---



---

**Algorithm 2** HiPPO

---

- 1: **Input:** skills  $\pi_{\theta_l}(a|s, z)$ , manager  $\pi_{\theta_h}(z|s)$ , horizon  $H$ , learning rate  $\alpha$
- 2: **while** not done **do**
- 3:   **for** actor = 1, 2, ..., N **do**
- 4:     Obtain trajectory with HiPPO Rollout
- 5:     Estimate advantages  $\hat{A}(a_{t'}, s_{t'}, z_t)$  and  $\hat{A}(z_t, s_t)$
- 6:   **end for**
- 7:    $\theta \leftarrow \theta + \alpha \nabla_{\theta} L_{HiPPO}^{CLIP}(\theta)$
- 8: **end while**

---

the KL-divergence between the old policy and the new policy to prevent this issue [80]. Unfortunately, hierarchical policies are generally represented by complex distributions without closed form expressions for the KL-divergence. Therefore, to improve the stability of our hierarchical policy gradient we turn towards Proximal Policy Optimization (PPO) [79]. PPO is a more flexible and compute-efficient algorithm. In a nutshell, it replaces the KL-divergence constraint with a cost function that achieves the same trust region benefits, but only requires the computation of the likelihood. Letting  $w_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ , the PPO objective is:

$$L^{CLIP}(\theta) = \mathbb{E}_t \min \{w_t(\theta)A_t, \text{clip}(w_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t\}$$

We can adapt our approximated hierarchical policy gradient with the same approach by letting  $w_{h,kp}(\theta) = \frac{\pi_{\theta_h}(z_{kp}|s_{kp})}{\pi_{\theta_{h,old}}(z_{kp}|s_{kp})}$  and  $w_{l,t}(\theta) = \frac{\pi_{\theta_l}(a_t|s_t, z_{kp})}{\pi_{\theta_{l,old}}(a_t|s_t, z_{kp})}$ , and using the super-index  $\text{clip}$  to denote the clipped objective version, we obtain the new surrogate objective:

$$L_{HiPPO}^{CLIP}(\theta) = \mathbb{E}_{\tau} \left[ \sum_{k=0}^{H/p} \min \{w_{h,kp}(\theta)A(s_{kp}, z_{kp}), w_{h,kp}^{\text{clip}}(\theta)A(s_{kp}, z_{kp})\} \right. \\ \left. + \sum_{t=0}^H \min \{w_{l,t}(\theta)A(s_t, a_t, z_{kp}), w_{l,t}^{\text{clip}}(\theta)A(s_t, a_t, z_{kp})\} \right]$$

We call this algorithm Hierarchical Proximal Policy Optimization (HiPPO). Next, we introduce a critical additions: a switching of the time-commitment between skills.



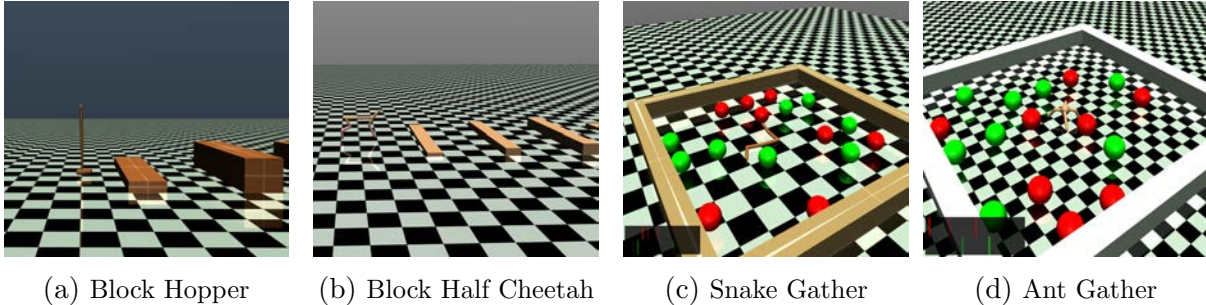


Figure 2.2: Environments used to evaluate the performance of our method. Every episode has a different configuration: wall heights for (a)-(b) and ball positions for (c)-(d)

## Varying Time-commitment

Most hierarchical methods either consider a fixed time-commitment to the lower level skills [19, 22], or implement the complex options framework [70, 5]. In this work we propose an in-between, where the time-commitment to the skills is a random variable sampled from a fixed distribution  $\text{Categorical}(T_{\min}, T_{\max})$  just before the manager takes a decision. This modification does not hinder final performance, and we show it improves zero-shot adaptation to a new task. This approach to sampling rollouts is detailed in Algorithm 1. The full algorithm is detailed in Algorithm 2.

## 2.6 Experiments

We designed our experiments to answer the following questions:

1. How does HiPPO compare against a flat policy when learning from scratch?
2. Does it lead to policies more robust to environment changes?
3. How well does it adapt already learned skills?
4. Does our skill diversity assumption hold in practice?

### Tasks

We evaluate our approach on a variety of robotic locomotion and navigation tasks. The Block environments, depicted in Fig. 2.2a-2.2b, have walls of random heights at regular intervals, and the objective is to learn a gait for the Hopper and Half-Cheetah robots to jump over them. The agents observe the height of the wall ahead and their proprioceptive information (joint positions and velocities), receiving a reward of +1 for each wall cleared. Hopper is a 3-link robot with a 14-dimensional observation space and a 3-dimensional action space.



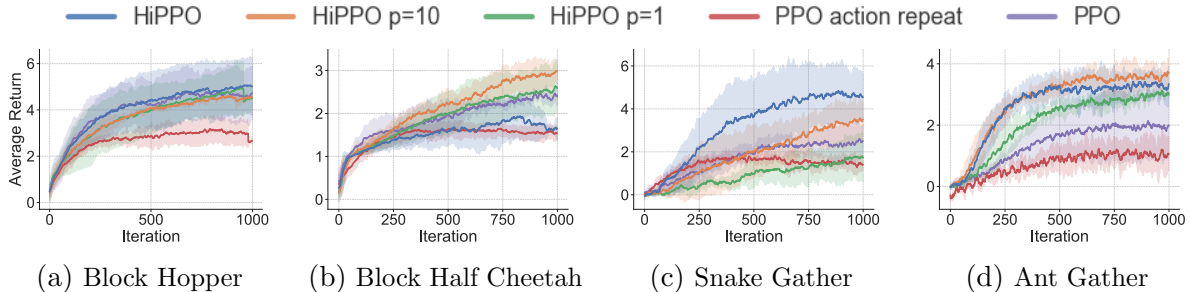


Figure 2.3: Analysis of different time-commitment strategies on learning from scratch.

Half-Cheetah has a 20-dimensional observation space and a 6-dimensional action space. We evaluate both of these agents on a sparse block hopping task. In addition to observing their own joint angles and positions, they observe the height and length of the next wall, the x-position of the next wall, and the distance to the wall from the agent. We also provide the same wall observations for the previous wall, which the agent can still interact with.

The Gather environments, described by Duan et al. [16], require agents to collect apples (green balls, +1 reward) while avoiding bombs (red balls, -1 reward). The only available perception beyond proprioception is through a LIDAR-type sensor indicating at what distance are the objects in different directions, and their type, as depicted in the bottom left corner of Fig. 2.2c-2.2d. This is challenging hierarchical task with sparse rewards that requires simultaneously learning perception, locomotion, and higher-level planning capabilities. Snake is a 5-link robot with a 17-dimensional observation space and a 4-dimensional action space. Ant is a quadrupedal robot with a 27-dimensional observation space and a 8-dimensional action space. Both Ant and Snake can move and rotate in all directions, and Ant faces the added challenge of avoiding falling over irrecoverably. In the Gather environment, agents also receive 2 sets of 10-dimensional lidar observations, which correspond to separate apple and bomb observations. The observation displays the distance to the nearest apple or bomb in each 36° bin, respectively. All environments are simulated with the physics engine MuJoCo [95].

## Learning from Scratch and Time-Commitment

In this section, we study the benefit of using our HiPPO algorithm instead of standard PPO on a flat policy [79]. The results, reported in Figure 2.3, demonstrate that training from scratch with HiPPO leads to faster learning and better performance than flat PPO. Furthermore, we show that the benefit of HiPPO does not just come from having temporally correlated exploration: PPO with action repeat converges at a lower performance than our method. HiPPO leverages the time-commitment more efficiently, as suggested by the poor performance of the ablation where we set  $p = 1$ , when the manager takes an action every environment step as well. Finally, Figure 2.4 shows the effectiveness of using the presented skill-dependent baseline.

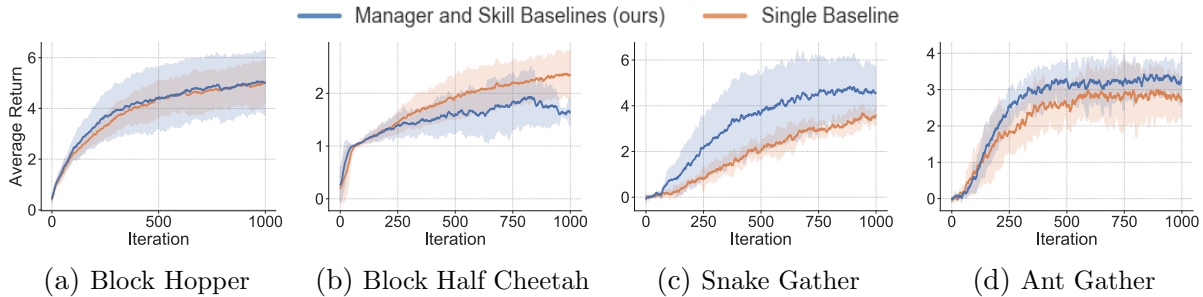


Figure 2.4: Using a skill-conditioned baseline, as defined in Section 2.5, generally improves performance of HiPPO when learning from scratch.

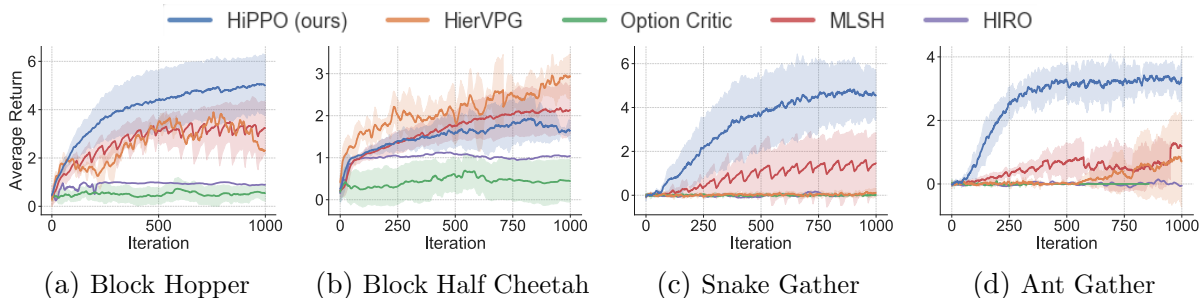


Figure 2.5: Comparison of HiPPO and HierVPG to prior hierarchical methods on learning from scratch.

### Comparison to Other Methods

We compare HiPPO to current state-of-the-art hierarchical methods. First, we evaluate HIRO [56], an off-policy RL method based on training a goal-reaching lower level policy. Fig. 2.5 shows that HIRO achieves poor performance on our tasks. As further detailed in Appendix 2.6, this algorithm is sensitive to access to ground-truth information, like the exact  $(x, y)$  position of the robot in Gather. In contrast, our method is able to perform well directly from the raw sensory inputs described in Section 2.6. We evaluate Option-Critic [5], a variant of the options framework [90] that can be used for continuous action-spaces. It fails to learn, and we hypothesize that their algorithm provides less time-correlated exploration and learns less diverse skills. We also compare against MLSH [22], which repeatedly samples new environment configurations to learn primitive skills. We take these hyperparameters from their Ant Twowalk experiment: resetting the environment configuration every 60 iterations, a warmup period of 20 during which only the manager is trained, and a joint training period of 40 during which both manager and skills are trained. Our results show that such a training scheme does not provide any benefits. Finally, we provide a comparison to a direct application of our Hierarchical Vanilla Policy Gradient (HierVPG) algorithm, and we see that the algorithm is unstable without PPO’s trust-region-like technique.

## Robustness to Dynamics Perturbations

We investigate the robustness of HiPPO to changes in the dynamics of the environment. We perform several modifications to the base Snake Gather and Ant Gather environments. One at a time, we change the body mass, dampening of the joints, body inertia, and friction characteristics of both robots. The results, presented in Table 2.1, show that HiPPO with randomized period `Categorical`( $[T_{\min}, T_{\max}]$ ) is able to better handle these dynamics changes. In terms of the drop in policy performance between the training environment and test environment, it outperforms HiPPO with fixed period on 6 out of 8 related tasks. These results suggest that the randomized period exposes the policy to a wide range of scenarios, which makes it easier to adapt when the environment changes.

Gather	Algorithm	Initial	Mass	Dampening	Inertia	Friction
Snake	Flat PPO	2.72	3.16 (+16%)	2.75 (+1%)	2.11 (-22%)	2.75 (+1%)
	HiPPO, $p = 10$	4.38	3.28 (-25%)	3.27 (-25%)	3.03 (-31%)	3.27 (-25%)
	HiPPO random $p$	5.11	<b>4.09</b> (-20%)	<b>4.03</b> (-21%)	<b>3.21</b> (-37%)	<b>4.03</b> (-21%)
Ant	Flat PPO	2.25	2.53 (+12%)	2.13 (-5%)	2.36 (+5%)	1.96 (-13%)
	HiPPO, $p = 10$	3.84	3.31 (-14%)	<b>3.37</b> (-12%)	2.88 (-25%)	<b>3.07</b> (-20%)
	HiPPO random $p$	3.22	<b>3.37</b> (+5%)	2.57 (-20%)	<b>3.36</b> (+4%)	2.84 (-12%)

Table 2.1: Zero-shot transfer performance. The final return in the initial environment is shown, as well as the average return over 25 rollouts in each new modified environment.

## Adaptation of Pre-Trained Skills

For the Block task, we use DIAYN [17] to train 6 differentiated subpolicies in an environment without any walls. Here, we see if these diverse skills can improve performance on a downstream task that’s out of the training distribution. For Gather, we take 6 pretrained subpolicies encoded by a Stochastic Neural Network [92] that was trained in a diversity-promoting environment [19]. We fine-tune them with HiPPO on the Gather environment, but with an extra penalty on the velocity of the Center of Mass. This can be understood as a preference for cautious behavior. This requires adjustment of the sub-policies, which were trained with a proxy reward encouraging them to move as far as possible (and hence quickly). Fig. 2.6 shows that using HiPPO to simultaneously train a manager and fine-tune the skills achieves higher final performance than fixing the sub-policies and only training a manager with PPO. The two initially learn at the same rate, but HiPPO’s ability to adjust to the new dynamics allows it to reach a higher final performance. Fig. 2.6 also shows that HiPPO can fine-tune the same given skills better than Option-Critic [5], MLSH [22], and HIRO [56].

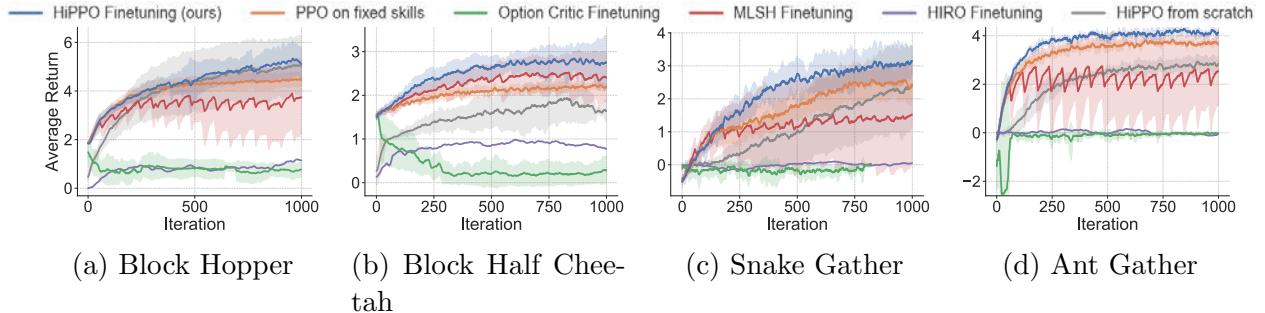


Figure 2.6: Benefit of adapting some given skills when the preferences of the environment are different from those of the environment where the skills were originally trained. Adapting skills with HiPPO has better learning performance than leaving the skills fixed or learning from scratch.

### Skill Diversity Assumption

In Lemma 1, we derived a more efficient and numerically stable gradient by assuming that the sub-policies are diverse. In this section, we empirically test the validity of our assumption and the quality of our approximation. We run the HiPPO algorithm on Ant Gather and Snake Gather both from scratch and with given pretrained skills, as done in the previous section. In Table 2.2, we report the average maximum probability under other sub-policies, corresponding to  $\epsilon$  from the assumption. In all settings, this is on the order of magnitude of 0.1. Therefore, under the  $p \approx 10$  that we use in our experiments, the term we neglect has a factor  $\epsilon^{p-1} = 10^{-10}$ . It is not surprising then that the average cosine similarity between the full gradient and our approximation is almost 1, as reported in Table 2.2.

Gather	Algorithm	Cosine Sim.	$\max_{z' \neq z_{kp}} \pi_{\theta_t}(a_t   s_t, z')$	$\pi_{\theta_t}(a_t   s_t, z_{kp})$
Snake	HiPPO on given skills	$0.98 \pm 0.01$	$0.09 \pm 0.04$	$0.44 \pm 0.03$
	HiPPO on random skills	$0.97 \pm 0.03$	$0.12 \pm 0.03$	$0.32 \pm 0.04$
Ant	HiPPO on given skills	$0.96 \pm 0.04$	$0.11 \pm 0.05$	$0.40 \pm 0.08$
	HiPPO on random skills	$0.94 \pm 0.03$	$0.13 \pm 0.05$	$0.31 \pm 0.09$

Table 2.2: Empirical evaluation of Lemma 1. In the middle and right columns, we evaluate the quality of our assumption by computing the largest probability of a certain action under other skills ( $\epsilon$ ), and the action probability under the actual latent. We also report the cosine similarity between our approximate gradient and the exact gradient from Eq. 2.3. The mean and standard deviation of these values are computed over the full batch collected at iteration 10.

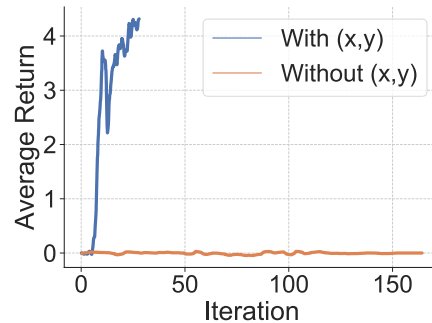


Figure 2.7: HIRO performance on Ant Gather with and without access to the ground truth  $(x, y)$ , which it needs to communicate useful goals.

## HIRO Sensitivity to Observation Space

In this section, we discuss why HIRO [56] performs poorly under our environments. As explained in our related work section, HIRO belongs to the general category of algorithms that train goal-reaching policies as lower levels of the hierarchy [98, 44]. These methods rely on having a goal-space that is meaningful for the task at hand. For example, in navigation tasks they require having access to the  $(x, y)$  position of the agent such that deltas in that space can be given as meaningful goals to move in the environment. Unfortunately, in many cases the only readily available information (if there’s no GPS signal or other positioning system installed) are raw sensory inputs, like cameras or the LIDAR sensors we mimic in our environments. In such cases, our method still performs well because it doesn’t rely on the goal-reaching extra supervision that is leveraged (and detrimental in this case) in HIRO and similar methods. In Figure 2.7, we show that knowing the ground truth location is critical for its success. We have reproduced the HIRO results in Fig. 2.7 using the published codebase, so we are convinced that our results showcase a failure mode of HIRO.

## 2.7 Conclusions and Future Work

In this work, we examined how to effectively adapt temporal hierarchies. We began by deriving a hierarchical policy gradient and its approximation. We then proposed a new method, HiPPO, that can stably train multiple layers of a hierarchy jointly. The adaptation experiments suggest that we can optimize pretrained skills for downstream environments, and learn emergent skills without any unsupervised pre-training. We also demonstrate that HiPPO with randomized period can learn from scratch on sparse-reward and long time horizon tasks, while outperforming non-hierarchical methods on zero-shot transfer.

# Chapter 3

## Generalized Hindsight for Reinforcement Learning

### 3.1 Acknowledgements

The work in this section was conducted, written up, and shared publicly by the author, Lerrel Pinto, and Pieter Abbeel [46].

### 3.2 Introduction

One key hallmark of biological learning is the ability to learn from mistakes. In RL, mistakes made while solving a task are only used to guide the learning of that particular task. But data seen while making these mistakes often contain a lot more information. In fact, extracting and re-using this information lies at the heart of most efficient RL algorithms. Model-based RL re-uses this information to learn a dynamics model of the environment. However for several domains, learning a robust model is often more difficult than directly learning the policy [16], and addressing this challenge continues to remain an active area of research [57]. Another way to re-use low-reward data is off-policy RL, where in contrast to on-policy RL, data collected from an older policy is re-used while optimizing the new policy. But in the context of multi-task learning, this is still inefficient (Section 3.5) since data generated from one task cannot effectively inform a different task. Towards solving this problem, recent work [3] focus on extracting even more information through *hindsight*.

In goal-conditioned settings, where tasks are defined by a sparse goal, HER [3] relabels the desired goal, for which a trajectory was generated, to a state seen in that trajectory. Therefore, if the goal-conditioned policy erroneously reaches an incorrect goal instead of the desired goal, we can re-use this data to teach it how to reach this incorrect goal. Hence, a low-reward trajectory under one desired goal is converted to a high-reward trajectory for the unintended goal. This new relabelling provides a strong supervision and produces significantly faster learning. However, a key assumption made in this framework is that

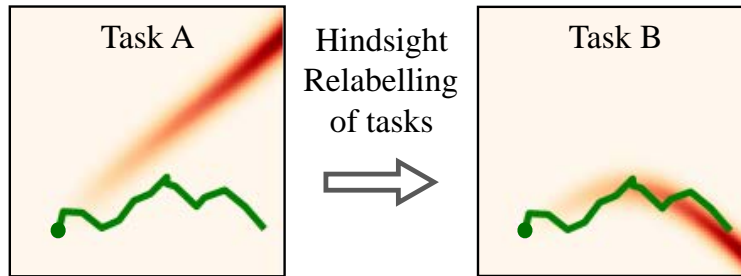


Figure 3.1: A rollout can often provide very little information about how to perform a task A. In the trajectory-following task on the left, the trajectory (green) sees almost no reward signal (areas in red). However, in the multi-task setting where each target trajectory represents a different task, we can find another task B for which our trajectory is a “pseudo-demonstration.” This hindsight relabelling provides high reward signal and enables sample-efficient learning.

goals are a sparse set of states that need to be reached. This allows for efficient relabelling by simply setting the relabeled goals to the states visited by the policy. But for several real world problems like energy-efficient transport, or robotic trajectory tracking, rewards are often complex combinations of desirables rather than sparse objectives. So how do we use hindsight for general families of reward functions?

In this paper, we build on the ideas of goal-conditioned hindsight and propose **Generalized Hindsight**. Here, instead of performing hindsight on a task-family of sparse goals, we perform hindsight on a task-family of reward functions. Since dense reward functions can capture a richer task specification, GH allows for better re-utilization of data. Note that this is done along with solving the task distribution induced by the family of reward functions. However for relabeling, instead of simply setting visited states as goals, we now need to compute the reward functions that best explain the generated data. To do this, we draw connections from Inverse Reinforcement Learning (IRL), and propose an approximate IRL relabeling algorithm we call AIR. Concretely, AIR takes a new trajectory and compares it to  $K$  randomly sampled tasks from our distribution. It selects the task for which the trajectory is a “pseudo-demonstration,” i.e. the trajectory achieves higher performance on that task than any of our previous trajectories. This “pseudo-demonstration” can then be used to quickly learn how to perform that new task. We go into detail on good selection algorithms in Section 3.5, and show an illustrative example of the relabeling process in Figure 3.1. We test our algorithm on several multi-task control tasks, and find that AIR consistently achieves higher asymptotic performance using as few as 20% of the environment interactions as our baselines. We also introduce a computationally more efficient version that also achieves higher asymptotic performance than our baselines.

In summary, we present three key contributions in this paper: (a) we extend the ideas of hindsight to the generalized reward family setting; (b) we propose AIR, a relabeling algorithm



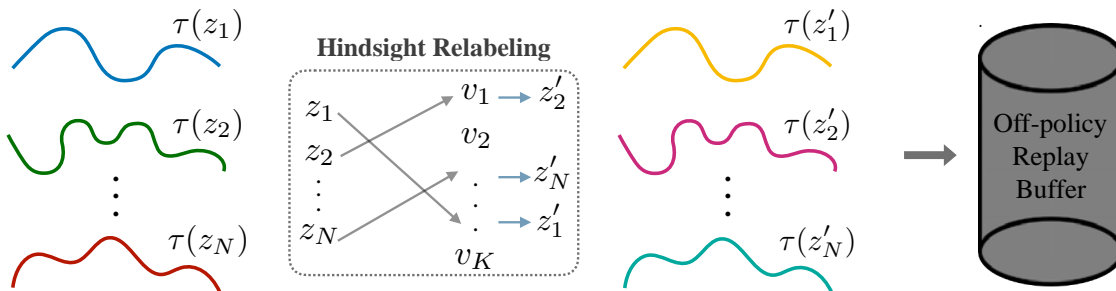


Figure 3.2: Trajectories  $\tau(z_i)$ , collected trying to maximize  $r(\cdot|z_i)$ , may contain very little reward signal about how to solve their original tasks. Generalized Hindsight checks against randomly sampled “candidate tasks”  $\{v_i\}_{i=1}^K$  to find different tasks  $z'_i$  for which these trajectories are “pseudo-demonstrations.” Using off-policy RL, we can obtain more reward signal from these relabeled trajectories.

using insights from IRL; and (c) we demonstrate significant improvements in multi-task RL on a suite of multi-task navigation and manipulation tasks.

### 3.3 Background

Before discussing our method, we briefly introduce some background and formalism for the RL algorithms used. A more comprehensive introduction to RL can be found in Kaelbling, Littman, and Moore [35] and Sutton and Barto [89].

#### Reinforcement Learning

In this work we deal with continuous space Markov Decision Processes  $\mathcal{M}$  that can be represented as the tuple  $\mathcal{M} \equiv (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mathbb{S})$ , where  $\mathcal{S}$  is a set of continuous states and  $\mathcal{A}$  is a set of continuous actions,  $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  is the transition probability function,  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward function,  $\gamma$  is the discount factor, and  $\mathbb{S}$  is the initial state distribution.

An episode for the agent begins with sampling  $s_0$  from the initial state distribution  $\mathbb{S}$ . At every timestep  $t$ , the agent takes an action  $a_t = \pi(s_t)$  according to a policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ . At every timestep  $t$ , the agent gets a reward  $r_t = r(s_t, a_t)$ , and the state transitions to  $s_{t+1}$ , which is sampled according to probabilities  $\mathcal{P}(s_{t+1}|s_t, a_t)$ . The goal of the agent is to maximize the expected return  $\mathbb{E}_{\mathbb{S}}[R_0|\mathbb{S}]$ , where the return is the discounted sum of the future rewards  $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$ . The  $Q$ -function is defined as  $Q^\pi(s_t, a_t) = E[R_t|s_t, a_t]$ . In the partial observability case, the agent takes actions based on the partial observation,  $a_t = \pi(o_t)$ , where  $o_t$  is the observation corresponding to the full state  $s_t$ .



## Off Policy RL using Soft Actor Critic

Generalized Hindsight requires an off-policy RL algorithm to perform relabeling. One popular off-policy algorithm for learning deterministic continuous action policies is Deep Deterministic Policy Gradients (DDPG) [50]. The algorithm maintains two neural networks: the policy (also called the actor)  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  (with neural network parameters  $\theta$ ) and a  $Q$ -function approximator (also called the critic)  $Q_\phi^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  (with neural network parameters  $\phi$ ).

During training, episodes are generated using a noisy version of the policy (called behaviour policy), e.g.  $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$ , where  $\mathcal{N}$  is the Normal distribution noise. The transition tuples  $(s_t, a_t, r_t, s_{t+1})$  encountered during training are stored in a replay buffer [53]. Training examples sampled from the replay buffer are used to optimize the critic. By minimizing the Bellman error loss  $\mathcal{L}_c = (Q(s_t, a_t) - y_t)^2$ , where  $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$ , the critic is optimized to approximate the  $Q$ -function. The actor is optimized by minimizing the loss  $\mathcal{L}_a = -\mathbb{E}_s[Q(s, \pi(s))]$ . The gradient of  $\mathcal{L}_a$  with respect to the actor parameters is called the deterministic policy gradient [85] and can be computed by backpropagating through the combined critic and actor networks. To stabilize the training, the targets for the actor and the critic  $y_t$  are computed on separate versions of the actor and critic networks, which change at a slower rate than the main networks. A common practice is to use a Polyak averaged [69] version of the main network. Soft Actor Critic (SAC) [27] builds on DDPG by adding an entropy maximization term in the reward. Since this encourages exploration and empirically performs better than most actor-critic algorithms, we use SAC for our experiments, although Generalized Hindsight is compatible with any off-policy RL algorithm.

## Multi-Task RL

The goal in multi-task RL is to not just solve a single MDP  $\mathcal{M}$ , but to solve a distribution of MDPs  $\mathcal{M}(z)$ , where  $z$  is the task-specification drawn from the task distribution  $z \sim \mathcal{T}$ . Although  $z$  can parameterize different aspects of the MDP, we are specially interested in the different reward functions. Hence, our distribution of MDPs is now  $\mathcal{M}(z) \equiv (\mathcal{S}, \mathcal{A}, \mathcal{P}, r(\cdot|z), \gamma, \mathbb{S})$ . Thus, a different  $z$  implies a different reward function under the same dynamics  $\mathcal{P}$  and start state  $s_0$ . One may view this representation as a generalization of the goal-conditioned RL setting [77], where the reward family is restricted to  $r(s, a|z = g) = -d(s, z = g)$ . Here  $d$  represents the distance between the current state  $s$  and the desired goal  $g$ . In sparse goal-conditioned RL, where hindsight has previously been applied [3], the reward family is further restricted to  $r(s, a|z = g) = [d(s, z = g) < \epsilon]$ . Here the agent gets a positive reward only when  $s$  is within  $\epsilon$  of the desired goal  $g$ .

## Hindsight Experience Replay (HER)

HER [3] is a simple method of manipulating the replay buffer used in off-policy RL algorithms that allows it to learn state-reaching policies more efficiently with sparse rewards. After

experiencing some episode  $s_0, s_1, \dots, s_T$ , every transition  $s_t \rightarrow s_{t+1}$  along with the goal for this episode is usually stored in the replay buffer. However with HER, the experienced transitions are also stored in the replay buffer with different goals. These additional goals are states that were achieved later in the episode. Since the goal being pursued does not influence the environment dynamics, one can replay each trajectory using arbitrary goals, assuming we use an off-policy RL algorithm to optimize [71].

## Inverse Reinforcement Learning (IRL)

In IRL [59], given an expert policy  $\pi_E$  or more practically, access to demonstrations  $\tau_E$  from  $\pi_E$ , we want to recover the underlying reward function  $r^*$  that best explains the expert behaviour. Although there are several methods that tackle this problem [73, 1, 104], the basic principle is to find  $r^*$  such that:

$$\mathbb{E}\left[\sum_{t=0}^{T-1} \gamma r^*(s_t) | \pi_E\right] \geq \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma r^*(s_t) | \pi\right] \quad \forall \pi \quad (3.1)$$

We use the framework of IRL to guide our *Approximate IRL* relabeling strategy for Generalized Hindsight.

## 3.4 Generalized Hindsight

### Overview

Given a multi-task RL setup, i.e. a distribution of reward functions  $r(\cdot|z)$ , our goal is to maximize the expected reward across the task distribution  $z \sim \mathcal{T}$  through optimizing our policy  $\pi$ :

$$\mathbb{E}_{z \sim \mathcal{T}} [R(\pi|z)] \quad (3.2)$$

Here,  $R(\pi|z) = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t \sim \pi(s_t|z)|z)$  represents the cumulative discounted reward under the reward parameterization  $z$  and the conditional policy  $\pi(\cdot|z)$ . One approach to solving this problem would be the straightforward application of RL to train the  $z$ -conditional policy using the rewards from  $r(\cdot|z)$ . However, this fails to re-use the data collected under one task parameter  $z$  ( $s_t, a_t \sim \pi(\cdot|z)$ ) to a different parameter  $z'$ . In order to better use and share this data, we propose to use hindsight relabeling, which is detailed in Algorithm 3.

The core idea of hindsight relabeling is to convert the data generated from the policy under one task  $z$  to a different task. Given the relabeled task  $z' = \text{relabel}(\tau(\pi(\cdot|z)))$ , where  $\tau$  represents the trajectory induced by the policy  $\pi(\cdot|z)$ , the state transition tuple  $(s_t, a_t, r_t(\cdot|z), s_{t+1})$  is converted to the relabeled tuple  $(s_t, a_t, r_t(\cdot|z'), s_{t+1})$ . This relabeled tuple is then added to the replay buffer of an off-policy RL algorithm and trained as if the data generated from  $z$  was generated from  $z'$ . If relabeling is done efficiently, it will allow

for data that is sub-optimal under one reward specification  $z$ , to be used for the better relabeled specification  $z'$ . In the context of sparse goal-conditioned RL, where  $z$  corresponds to a goal  $g$  that needs to be achieved, HER [3] relabels the goal to states seen in the trajectory, i.e.  $g' \sim \tau(\pi(\cdot|z = g))$ . This labeling strategy, however, only works in sparse goal conditioned tasks. In the following section, we describe two relabeling strategies that allow for a generalized application of hindsight.

---

**Algorithm 3** Generalized Hindsight
 

---

- 1: **Input:** Off-policy RL algorithm  $\mathbb{A}$ , strategy  $\mathbb{S}$  for choosing suitable task variables to relabel with, reward function  $r : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow \mathbb{R}$
  - 2: **for** episode = 1 to  $M$  **do**
  - 3:   Sample a task variable  $z$  and an initial state  $s_0$
  - 4:   Roll out policy on  $z$  for  $T$  steps, yielding trajectory  $\tau$
  - 5:   Find set of new tasks to relabel with:  $Z := \mathbb{S}(\tau)$
  - 6:   Store original transitions in replay buffer:  
        $(s_t, a_t, r(s_t, a_t, z), s_{t+1}, z)$
  - 7:   **for**  $z' \in Z$  **do**
  - 8:     Store relabeled transitions in replay buffer:  
        $(s_t, a_t, r(s_t, a_t, z'), s_{t+1}, z')$
  - 9:   **end for**
  - 10:   Perform  $n$  steps of policy optimization with  $\mathbb{A}$
  - 11: **end for**
- 

## Approximate IRL Relabeling (AIR)

---

**Algorithm 4**  $\mathbb{S}_{IRL}$ : Approximate IRL
 

---

- 1: **Input:** Trajectory  $\tau = (s_0, a_0, \dots, s_T)$ , cached trajectories  $\mathcal{D} = \{(s_0, a_0, \dots, s_T)\}_{i=1}^N$ , reward function  $r : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow \mathbb{R}$ , number of candidate task variables to try:  $K$ , number of task variables to return:  $m$
  - 2: Sample set of candidate tasks  $Z = \{v_j\}_{j=1}^K$ , where  $v_j \sim \mathcal{T}$   
**Approximate IRL Strategy:**
  - 3: **for**  $v_j \in Z$  **do**
  - 4:   **Calculate trajectory reward** for  $\tau$  and the trajectories in  $\mathcal{D}$ :  $R(\tau|v_j) := \sum_{t=0}^T \gamma^t r(s_t, a_t, v_j)$
  - 5:   **Calculate percentile estimate:**  
        $\hat{P}(\tau, v_j) = \frac{1}{n} \sum_{i=1}^N \mathbb{1}\{R(\tau|v_j) \geq R(\tau_i|v_j)\}$
  - 6: **end for**
  - 7: **return**  $m$  tasks  $v_j$  with highest percentiles  $\hat{P}(\tau, v_j)$
-

The goal of computing the optimal reward parameter, given a trajectory is closely tied to the Inverse Reinforcement Learning (IRL) setting. In IRL, given demonstrations from an expert, we can retrieve the reward function the expert was optimized for. At the heart of these IRL algorithms, a reward specification parameter  $z'$  is optimized such that

$$R(\tau_E|z') \geq R(\tau'|z') \quad \forall \tau' \quad (3.3)$$

where  $\tau_E$  is an expert trajectory. Inspired by the IRL framework, we propose the *Approximate IRL* relabeling seen in Algorithm 4. We can use a buffer of past trajectories to find the task  $z'$  on which our current trajectory does better than the older ones. Intuitively this can be seen as an approximation of the right hand side of Eq. 3.3. Concretely, we want to relabel a new trajectory  $\tau$ , and have  $N$  previously sampled trajectories along with  $K$  randomly sampled candidate tasks  $v_k$ . Then, the relabeled task for trajectory  $\tau$  is computed as:

$$z' = \arg \max_k \frac{1}{N} \sum_{j=1}^N \mathbb{1}\{R(\tau|v_k) \geq R(\tau_j|v_k)\} \quad (3.4)$$

The relabeled  $z'$  for  $\tau$  maximizes its percentile among the  $N$  most recent trajectories collected with our policy. One can also see this as an approximation of max-margin IRL 73. One potential challenge with large  $K$  is that many  $v_k$  will have the same percentile. To choose between these potential task relabelings, we add tiebreaking based on the advantage estimate

$$\hat{A}(\tau, z) = R(\tau|z) - V^\pi(s_0, z) \quad (3.5)$$

Among candidate tasks  $v_k$  with the same percentile, we take the tasks that have higher advantage estimate. From here on, we will refer to Generalized Hindsight with Approximate IRL Relabeling as AIR.

## Advantage Relabeling

One potential problem with AIR is that it requires  $O(NT)$  time to compute the relabeled task variable for each new trajectory, where  $N$  is the number of past trajectories compared to, and  $T$  is the horizon. A relaxed version of AIR could significantly reduce computation time, while maintaining relatively high-accuracy relabeling. One way to do this is to use the *Maximum-Reward* relabeling objective. Instead of choosing from our  $K$  candidate tasks  $v_k \sim \mathcal{T}$  by selecting for high percentile (Equation 3.3), we could relabel based on the cumulative trajectory reward:

$$z' = \arg \max_{v_k} \{R(\tau|v_k)\} \quad (3.6)$$

However, one challenge with simply taking the *Maximum-Reward* relabel is that different reward parameterizations may have different scales which will bias the relabels to a specific  $z$ . Say for instance there exists a task in the reward family  $v_j$  such that  $r(\cdot|v_j) = 1 + \max_{i \neq j} r(\cdot|v_i)$ . Then,  $v_j$  will always be the relabeled reward parameter irrespective of the

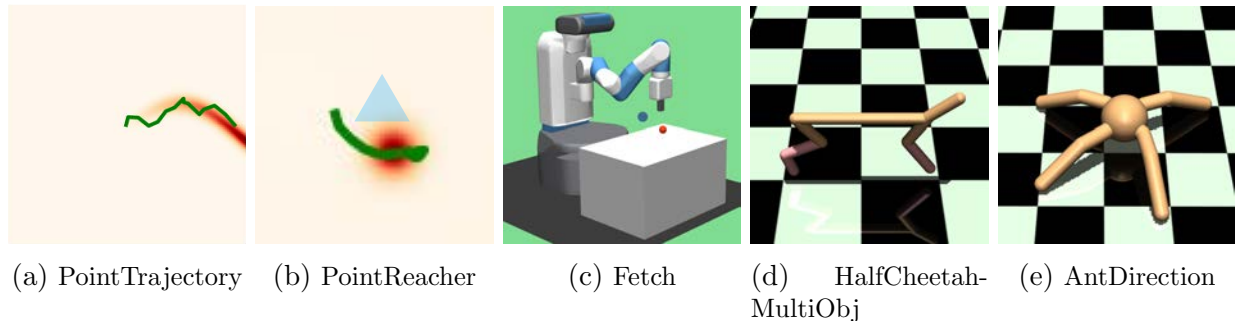


Figure 3.3: Environments we report comparisons on. PointTrajectory requires a 2D pointmass to follow a target trajectory; PointReacher requires moving the pointmass to a goal location, while avoiding an obstacle and modulating its energy usage. In (b), the red circle indicates the goal location, while the blue triangle indicates an imagined obstacle to avoid. Fetch has the same reward formulation as PointReacher, but requires controlling the noisy Fetch robot in 3 dimensions. HalfCheetah requires learning running in both directions, flipping, jumping, and moving efficiently. AntDirection requires moving in a target direction as fast as possible.

trajectory  $\tau$ . Hence, we should not only care about the  $v_k$  that maximizes reward, but select  $v_k$  such that  $\tau$ 's likelihood under the trajectory distribution drawn from the optimal  $\pi^*(\cdot|v_k)$  is high. To do this, we can simply select  $z'$  based on the advantage term that we used to tiebreak for AIR.

$$z'_i = \arg \max_k R(\tau|v_k) - V^\pi(s_0, v_k) \quad (3.7)$$

We call this *Advantage* relabeling (Algorithm 5), a more efficient, albeit less accurate, version of AIR. Empirically, *Advantage* relabeling often performs as well as AIR, but requires the value function  $V^\pi$  to be more accurate than it has to be in AIR. We reuse the twin  $Q$ -networks from SAC as our value function.

$$V^\pi(s, z) = \min(Q_1(s, \pi(s|z), z), Q_2(s, \pi(s|z), z)) \quad (3.8)$$

## 3.5 Experimental Evaluation

In this section, we describe our environment settings along with a discussion of our central hypothesis: Does relabeling improve performance?

### Environments

Multi-task RL with a generalized family of reward parameterizations does not have existing benchmark environments. However, since sparse goal-conditioned RL has benchmark envi-

---

**Algorithm 5**  $\mathbb{S}_A$ : Trajectory Advantage

---

- 1: Repeat steps 1 & 2 from Algorithm 4
  - Advantage Relabeling Strategy:**
  - 2: **for**  $v_j \in Z$  **do**
  - 3:   **Calculate trajectory reward:**  

$$R(\tau|v_j) := \sum_{t=0}^T \gamma^t r(s_t, a_t, v_j)$$
  - 4:   **Calculate advantage estimate of the trajectory:**  

$$\hat{A}(\tau, v_j) = R(\tau|v_j) - V^\pi(s_0, v_j)$$
  - 5: **end for**
  - 6: **return**  $m$  tasks  $z_j$  with highest advantages  $\hat{A}(\tau, z_j)$
- 

ronments [68], we build on their robotic manipulation framework to make our environments. The key difference in the environment setting between ours and Plappert et al. [68] is that in addition to goal reaching, we have a dense reward parameterization for practical aspects of manipulation like energy consumption [51] and safety [10]. These environments will be released for open-source access. The five environments we use are as follows:

1. **PointTrajectory**: 2D pointmass with  $(x, y)$  observations and  $(dx, dy)$  position control for actions. The goal is to follow a target trajectory parameterized by  $z \in \mathcal{Z} \subseteq \mathbb{R}^3$ . Figure 3.3a depicts an example trajectory in green, overlaid on the reward heatmap defined by some specific task  $z$ .
2. **PointReacher**: 2D pointmass with  $(x, y)$  observations and  $(dx, dy)$  position control for actions. This environment has high reward around the goal position  $(x_g, y_g)$  and low reward around an obstacle location  $(x_{obst}, y_{obst})$ . The 6-dimensional task vector is  $z = (x_g, y_g, x_{obst}, y_{obst}, u, v)$ , where  $u$  and  $v$  control the weighting between the goal rewards, obstacle rewards, and action magnitude penalty.
3. **Fetch**: Here we adapt the Fetch environment from OpenAI Gym [8], with  $(x, y, z)$  end-effector position as observations and noisy position control for actions. We use the same parameterized reward function as in PointReacher that includes energy and safety specifications.
4. **HalfCheetahMultiObjective**: HalfCheetah-V2 from OpenAI Gym, with 17-dimensional observations and 6-dimensional actions for torque control. The task variable  $z = (w_{vel}, w_{rot}, w_{height}, w_{energy}) \in \mathcal{Z} = S^3$  controls the weights on the forward velocity, rotation speed, height, and energy rewards.
5. **AntDirection**: Ant-V2 from OpenAI gym, with 111-dimensional observations and 8-dimensional actions for torque control. The task variable  $z \in [-180^\circ, +180^\circ]$  parameterizes the target direction. The reward function is:

$$r(\cdot|z) = \|\text{velocity}\|_2 \times \mathbb{1}\{\text{velocity angle within 15 degrees of } z\}$$

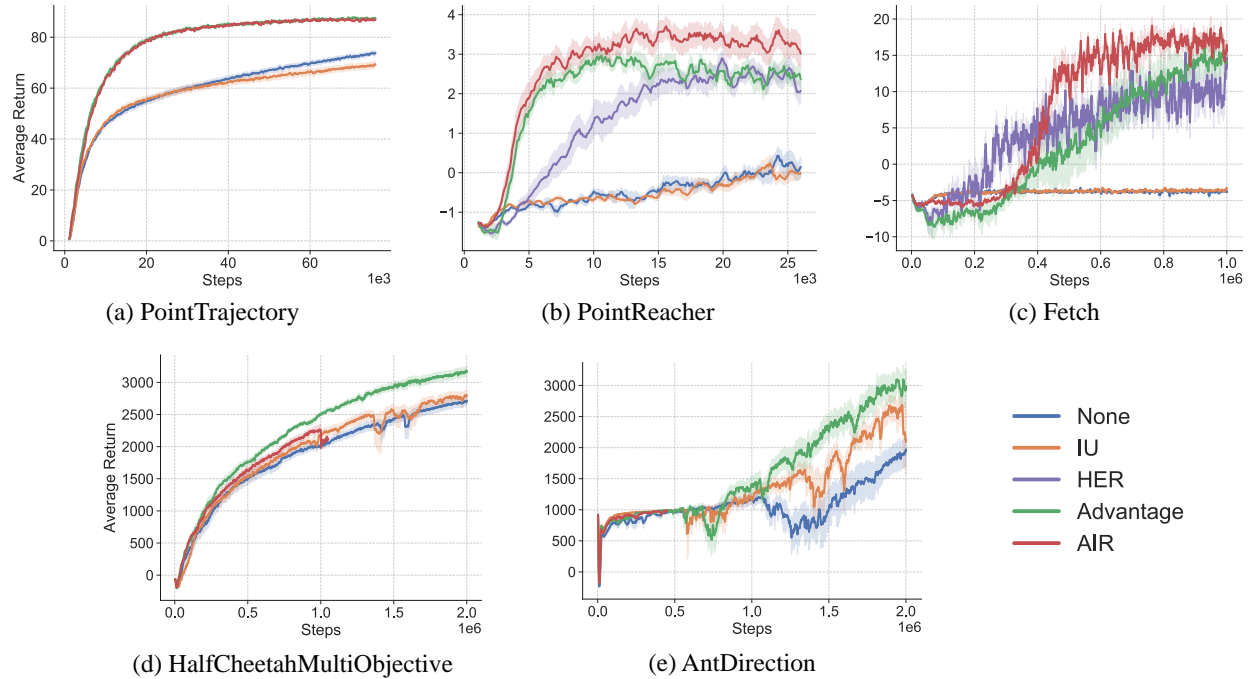


Figure 3.4: Learning curves comparing Generalized Hindsight algorithms to baseline methods. For environments with a goal-reaching component, we also compare to HER. In (a), AIR learning curve obscures the Advantage learning curve. In (d) and (e), where we use  $N = 500$  for AIR, AIR takes much longer to run than the other methods. 10 seeds were used for all runs.

## Does Relabeling Help?

To understand the effects of relabeling, we compare our technique with the following standard baseline methods:

- No relabeling (None): as done in [103], we train with standard SAC without any relabeling step.
- Intentional-Unintentional Agent (IU) [9]: when there is only a finite number of tasks, IU relabels a trajectory with every task variable. Since our space of tasks is continuous, we relabel with random  $z' \sim \mathcal{T}$ . This allows for information to be shared across tasks, albeit in a more diluted form.
- HER: for goal-conditioned tasks, we use HER to relabel the goal portion of the latent with the future relabeling strategy.

We compare the learning performance for AIR and Advantage Relabeling with these baselines on our suite of environments in Figure 3.4. On all tasks, AIR and Advantage Relabeling out-



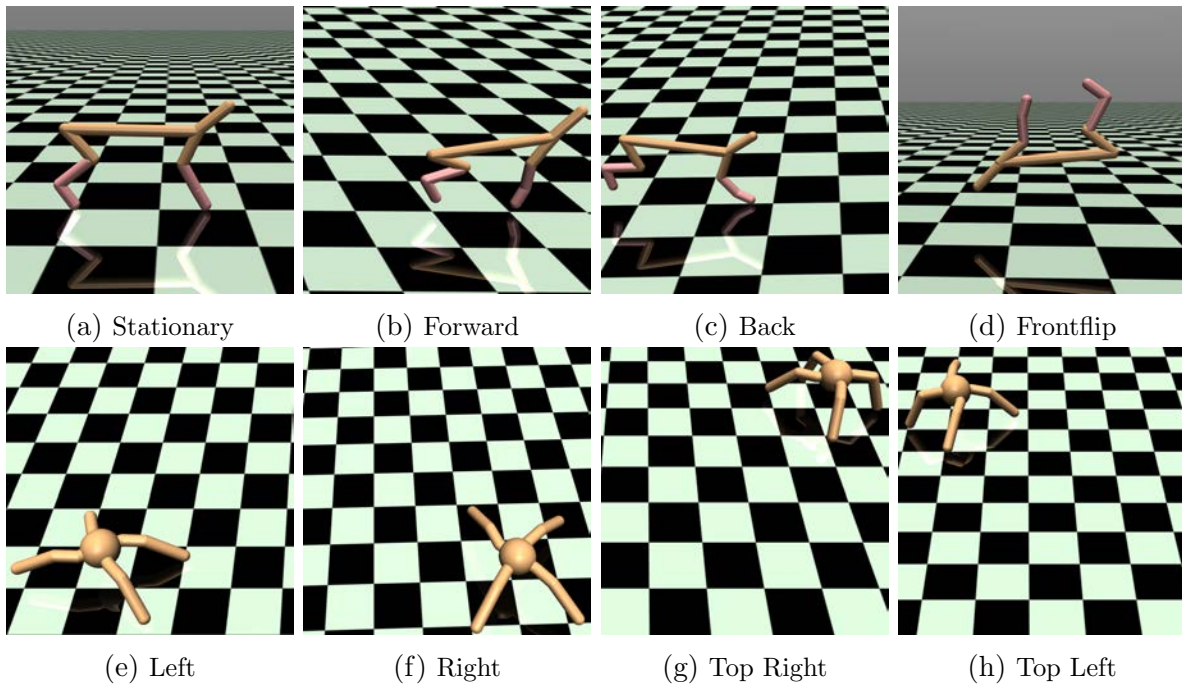


Figure 3.5: The agents efficiently learn a wide range of behaviors. On HalfCheetahMultiObjective, the robot can stay still to conserve energy, run quickly forwards and backwards, and do a frontflip. On AntDirection, the robot can run quickly in any given direction.

perform the baselines in both sample-efficiency and asymptotic performance. Both of our relabeling strategies outperform the Intentional-Unintentional Agent, implying that selectively relabeling trajectories with a few carefully chosen  $z'$  is more effective than relabeling with many random tasks. Collectively, these results show that AIR can greatly improve learning performance, even on highly dense environments such as HalfCheetahMultiObjective, where learning signal is readily available.

### How does generalized relabeling compare to HER?

HER is, by design, limited to goal-reaching environments. For environments such as HalfCheetahMultiObjective, HER cannot be applied to relabel the weights on velocity, rotation, height, and energy. However, we can compare AIR with HER on the partially goal-reaching environments PointReacher and Fetch. [Figure 3.4](#) shows that AIR achieves higher asymptotic performance than HER on both these environments. [Figure 3.6](#) demonstrates on PointReacher how AIR can better choose the non-goal-conditioned parts of the task. Both HER and AIR understand to place the relabeled goal around the terminus of the trajectory. However, only AIR understands that the imagined obstacle should be placed above the goal, since this trajectory becomes an optimal example of how to reach the new goal while avoiding



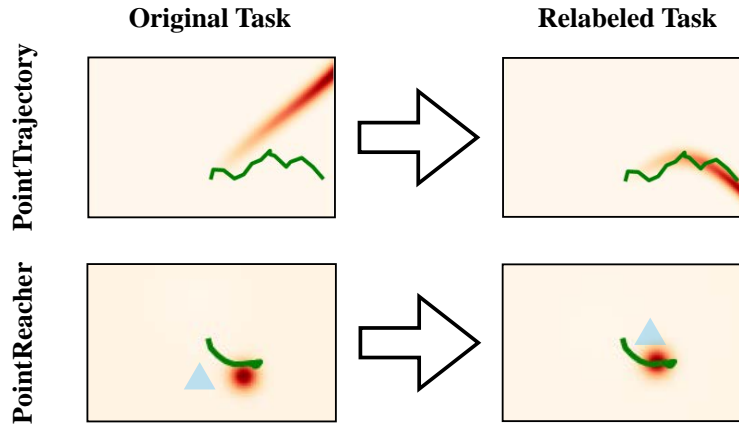


Figure 3.6: Red denotes areas of high reward, for following a target trajectory (top) or reaching a goal (bottom). Blue indicates areas of negative reward, where an obstacle may be placed. On both environments, relabeling finds tasks on which our trajectory has high reward signal. On PointReacher, AIR does not place the obstacle arbitrarily far. It places the relabeled obstacle within the curve of the trajectory, since this is the only way that the curved path would be better than a straight-line path (that would come close to the relabeled obstacle).

the obstacle. HER (as well as the Intentional-Unintentional Agent) offer no such specificity, either leaving the obstacle in place or randomly placing it.

### Analysis of Relabeling Fidelity

Approximate IRL, advantage relabeling, and reward relabeling are all approximate methods for finding the optimal task  $z^*$  that a trajectory is (close to) optimal for. As a result, an important characteristic is their *fidelity*, i.e. how close the  $z'$  they choose is to the true  $z^*$ . In Figure 3.7, we compare the fidelities of these three algorithms. Approximate IRL comes fairly close to reproducing the true  $z^*$ , albeit a bit noisily because it relies on the comparison to  $N$  past trajectories. Advantage relabeling is slightly more precise, but fails for large energy weights, likely because the value function is not precise enough to differentiate between these tasks. Finally, reward relabeling does poorly, since it naively assigns  $z'$  solely based on the trajectory reward, not how close the trajectory reward is to being optimal.

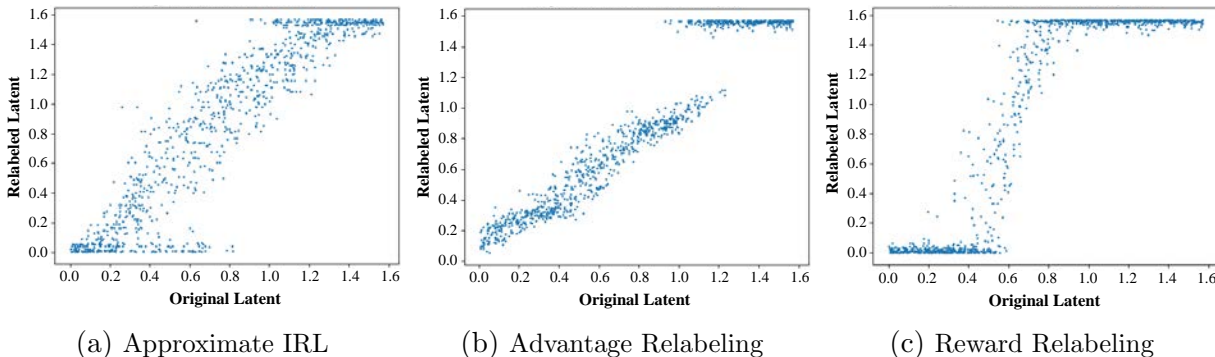


Figure 3.7: Comparison of relabeling fidelity on optimal trajectories for approximate IRL, advantage relabeling, and reward relabeling. We train a multi-task policy to convergence on the PointReacher environment. We roll out our policy on 1000 randomly sampled tasks  $z$ , and apply each relabeling method to select from  $K = 100$  randomly sampled tasks  $v$ . For approximate IRL, we compare against  $N = 10$  prior trajectories. The x-axis shows the weight on energy for the task  $z$  used for the rollout, while the y-axis shows the weight on energy for the relabeled task  $z'$ . Note that goal location, obstacle location, and weights on their rewards/penalties are varying as well, but are not shown. Closer to to the line  $y = x$  indicates higher fidelity, since it implies  $z' \approx z^*$ .

## 3.6 Related Work

### Multi-task and Transfer Learning

Learning models that can share information across tasks has been concretely studied in the context multi-task learning [11], where models for multiple tasks are simultaneously learned. More recently, Kokkinos [40] and Doersch and Zisserman [15] looks at shared learning across visual tasks, while Devin et al. [14] and Pinto and Gupta [66] looks at shared learning across robotic tasks.

Transfer learning [62, 96] focuses on transferring knowledge from one domain to another. One of the simplest forms of transfer is finetuning [24], where instead of learning a task from scratch it is initialized on a different task. Several other works look at more complex forms of transfer [102, 33, 4, 76, 41, 18, 25, 34].

In the context of RL, transfer learning [93] research has focused on learning transferable features across tasks [63, 6, 60]. Another line of work by [74, 37, 14] has focused on network architectures that improves transfer of RL policies. Another way of getting generalizable policies is through domain randomization [75, 94], i.e. train an unconditional policy across all of the domains in the multi-task learning setting. Although this works for task distributions over the dynamics and observation space [67], it cannot handle distributions of reward functions as seen in our experiments. The techniques of domain randomization are however complementary to our method, where it can provide generalizability to dynamics and ob-

ervation space while Generalized Hindsight can provide generalizability to different reward functions.

Hierarchical reinforcement learning [55, 7] is another framework amenable for multitask learning. Here the key idea is to have a hierarchy of controllers. One such setup is the Options framework [91] where the higher level controllers breaks down a task into sub-tasks and chooses a low-level controller to complete that sub-task. Unsupervised learning of these low-level controllers has been a focus of recent research [19, 17, 81]. Variants of the Options framework [22, 47] have examined how to train hierarchies in a multi-task setting, but information re-use across tasks remains restricted to learning transferable primitives. Generalized Hindsight could be used to train these hierarchical policies more efficiently.

## Hindsight in RL

Hindsight methods have been used for improving learning across a variety of applications. Andrychowicz et al. [3] uses hindsight to efficiently learn on sparse, goal-conditioned tasks. Nair et al. [58] approaches goal-reaching with visual input by learning a latent space encoding for images, and using hindsight relabeling within that latent space. Several hierarchical methods [44, 56] train a low-level policy to achieve subgoals and a higher-level controller to propose those subgoals. These methods use hindsight relabeling to help the higher-level learn, even when the low-level policy fails to achieve the desired subgoals. Generalized Hindsight could be used to allow for richer low-level reward functions, potentially allowing for more expressive hierarchical policies.

## Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) has had a rich history of solving challenging robotics problems [1, 59]. More recently, powerful function approximators have enabled more general purpose IRL. For instance, Ho and Ermon [32] use an adversarial framework to approximate the reward function. Li, Song, and Ermon [49] build on top of this idea by learning reward functions on demonstrations from a mixture of experts. Although, our relabeling strategies currently build on top of max-margin based IRL [73], our central idea is orthogonal to the choice of IRL techniques and can be combined with more complex function approximators.

## 3.7 Conclusions

In this work, we have presented Generalized Hindsight, an approximate IRL based task relabeling algorithm for multi-task RL. We demonstrate how efficient relabeling strategies can significantly improve performance on simulated navigation and manipulation tasks. Through these first steps, we believe that this technique can be extended to other domains like real world robotics, where a balance between different specifications, such as energy use or safety, is important.

# Chapter 4

## Conclusion

The methods we have introduced over the past two chapters represent small steps toward producing autonomous generalists that can quickly pick up and excel at new tasks. In Chapter [2](#), our exploration of modular policies led us to propose HiPPO, a novel on-policy hierarchical reinforcement learning algorithm. Supported by several theoretical results, HiPPO learns robust, transferable skills, and can quickly learn a new task by finetuning skills learned in other environments. In Chapter [3](#), we introduced *generalized hindsight*, which uses approximate inverse reinforcement learning to accelerate multi-task training. Generalized hindsight matches generated behaviors with the tasks they are best suited for, allowing for substantially faster learning from environment interaction.

Learning each problem individually “from scratch” remains as a common practice in research and industry. In a world where data, compute, time, and energy are limited resources, many useful applications remain out of reach if we stick to the single-task paradigm. Much work remains to be done before machine learning algorithms can learn as efficiently as humans can – when doing something new takes a handful of tries, not thousands. We hope that the methods presented in this work may serve as inspiration for future research that marches towards this frontier.

# Bibliography

- [1] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 1.
- [2] Jacob Andreas, Dan Klein, and Sergey Levine. “Modular Multitask Reinforcement Learning with Policy Sketches”. In: *International Conference in Machine Learning* (2017). URL: <http://github.com/>.
- [3] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *NIPS* (2017).
- [4] Yusuf Aytar and Andrew Zisserman. “Tabula rasa: Model transfer for object category detection”. In: *2011 international conference on computer vision*. IEEE. 2011, pp. 2252–2259.
- [5] Pierre-Luc Bacon, Jean Harb, and Doina Precup. “The Option-Critic Architecture”. In: *AAAI* (2017), pp. 1726–1734. URL: <http://arxiv.org/abs/1609.05140>.
- [6] André Barreto et al. “Successor features for transfer in reinforcement learning”. In: *Advances in neural information processing systems*. 2017, pp. 4055–4065.
- [7] Andrew G Barto, Satinder Singh, and Nuttapon Chentanez. “Intrinsically motivated learning of hierarchical collections of skills”. In: *Proceedings of the 3rd International Conference on Development and Learning*. 2004, pp. 112–19.
- [8] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).
- [9] Serkan Cabi et al. “The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously”. In: *arXiv preprint arXiv:1707.03300* (2017).
- [10] Sylvain Calinon, Irene Sardellitti, and Darwin G Caldwell. “Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 249–254.
- [11] Rich Caruana. “Multitask learning”. In: *Machine learning* 28.1 (1997), pp. 41–75.
- [12] Christian Daniel et al. “Probabilistic inference for determining options in reinforcement learning”. In: *Machine Learning* 104.104 (2016). DOI: [10.1007/s10994-016-5580-x](https://doi.org/10.1007/s10994-016-5580-x).

- [13] Peter Dayan and Geoffrey E. Hinton. “Feudal Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* (1993), pp. 271–278. ISSN: 0143991X. DOI: [10.1108/IR-08-2017-0143](https://doi.org/10.1108/IR-08-2017-0143), URL: <http://www.cs.toronto.edu/~fritz/absps/dh93.pdf>.
- [14] Coline Devin et al. “Learning modular neural network policies for multi-task and multi-robot transfer”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2169–2176.
- [15] Carl Doersch and Andrew Zisserman. “Multi-task self-supervised visual learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2051–2060.
- [16] Yan Duan et al. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *International Conference in Machine Learning* (2016). URL: <http://arxiv.org/abs/1604.06778>.
- [17] Benjamin Eysenbach et al. “Diversity is All You Need: Learning Skills without a Reward Function”. In: *International Conference in Learning Representations* (2019). URL: <http://arxiv.org/abs/1802.06070>.
- [18] Basura Fernando et al. “Unsupervised visual domain adaptation using subspace alignment”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2960–2967.
- [19] Carlos Florensa, Yan Duan, and Pieter Abbeel. “Stochastic Neural Networks for Hierarchical Reinforcement Learning”. In: *International Conference in Learning Representations* (2017), pp. 1–17. ISSN: 14779129. DOI: [10.1002/rcm.765](https://doi.org/10.1002/rcm.765). URL: <http://arxiv.org/abs/1704.03012>.
- [20] Carlos Florensa et al. “Reverse Curriculum Generation for Reinforcement Learning”. In: *Conference on Robot Learning* (2017), pp. 1–16. ISSN: 1938-7228. DOI: [10.1080/00908319208908727](https://doi.org/10.1080/00908319208908727). URL: <http://arxiv.org/abs/1707.05300>.
- [21] Carlos Florensa et al. “Self-supervised Learning of Image Embedding for Continuous Control”. In: *Workshop on Inference to Control at NeurIPS*. 2018. URL: <http://arxiv.org/abs/1901.00943>.
- [22] Kevin Frans et al. “META LEARNING SHARED HIERARCHIES”. In: *International Conference on Learning Representations*. 2018. URL: <https://openreview.net/forum?id=SyX0IeWAW>.
- [23] Mohammad Ghavamzadeh and Sridhar Mahadevan. “Hierarchical Policy Gradient Algorithms”. In: *International Conference in Machine Learning* (2003). URL: [http://chercheurs.lille.inria.fr/~ghavamza/my\\_website/Publications\\_files/icml03.pdf](http://chercheurs.lille.inria.fr/~ghavamza/my_website/Publications_files/icml03.pdf).
- [24] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

- [25] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. “Domain adaptation for object recognition: An unsupervised approach”. In: *2011 international conference on computer vision*. IEEE. 2011, pp. 999–1006.
- [26] Tuomas Haarnoja et al. “Latent Space Policies for Hierarchical Reinforcement Learning”. In: *International Conference in Machine Learning* (2018). URL: <http://arxiv.org/abs/1804.02808>.
- [27] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *arXiv preprint arXiv:1801.01290* (2018).
- [28] Jean Harb et al. “When Waiting is not an Option : Learning Options with a Deliberation Cost”. In: *AAAI* (Sept. 2017). URL: <http://arxiv.org/abs/1709.04571>.
- [29] Karol Hausman et al. “Learning an Embedding Space for Transferable Robot Skills”. In: *International Conference in Learning Representations* (2018), pp. 1–16.
- [30] Nicolas Heess et al. “Emergence of Locomotion Behaviours in Rich Environments”. In: (July 2017). URL: <http://arxiv.org/abs/1707.02286>.
- [31] Nicolas Heess et al. “Learning and Transfer of Modulated Locomotor Controllers”. In: (2016). URL: <https://arxiv.org/abs/1610.05182>.
- [32] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems*. 2016, pp. 4565–4573.
- [33] Judy Hoffman, Trevor Darrell, and Kate Saenko. “Continuous manifold based adaptation for evolving visual domains”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 867–874.
- [34] I-Hong Jhuo et al. “Robust visual domain adaptation with low-rank reconstruction”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 2168–2175.
- [35] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* (1996).
- [36] Sham Kakade. “A Natural Policy Gradient”. In: *Advances in Neural Information Processing Systems* (2002).
- [37] Ken Kanksy et al. “Schema networks: Zero-shot transfer with a generative causal model of intuitive physics”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1809–1818.
- [38] Avi Karni et al. “The acquisition of skilled motor performance: fast and slow experience-driven changes in primary motor cortex”. In: *Proceedings of the National Academy of Sciences* 95.3 (1998), pp. 861–868.
- [39] Elia Kaufmann et al. “Deep drone racing: Learning agile flight in dynamic environments”. In: *arXiv preprint arXiv:1806.08548* (2018).



- [40] Iasonas Kokkinos. “Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6129–6138.
- [41] Brian Kulis, Kate Saenko, and Trevor Darrell. “What you saw is not what you get: Domain adaptation using asymmetric kernel transforms”. In: *CVPR 2011*. IEEE. 2011, pp. 1785–1792.
- [42] Tejas D Kulkarni et al. “Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation”. In: *Advances in Neural Information Processing Systems* (2016), pp. 1–13.
- [43] Hoang M Le et al. “Hierarchical Imitation and Reinforcement Learning”. In: *International Conference in Machine Learning* (2018).
- [44] Andrew Levy, Robert Platt, and Kate Saenko. “Hierarchical Actor-Critic”. In: *arXiv:1712.00948* (Dec. 2017). URL: <http://arxiv.org/abs/1712.00948>.
- [45] Andrew Levy, Robert Platt, and Kate Saenko. “Hierarchical Reinforcement Learning with Hindsight”. In: *International Conference on Learning Representations* (May 2019). URL: <http://arxiv.org/abs/1805.08180>.
- [46] Alexander C Li, Lerrel Pinto, and Pieter Abbeel. “Generalized Hindsight for Reinforcement Learning”. In: *arXiv preprint arXiv:2002.11708* (2020).
- [47] Alexander C Li et al. “Sub-policy Adaptation for Hierarchical Reinforcement Learning”. In: *arXiv preprint arXiv:1906.05862* (2019).
- [48] Alexander Li et al. “Sub-policy Adaptation for Hierarchical Reinforcement Learning”. In: *International Conference on Learning Representations*. 2020. URL: <https://openreview.net/forum?id=ByeWogStDS>.
- [49] Yunzhu Li, Jiaming Song, and Stefano Ermon. “Infogail: Interpretable imitation learning from visual demonstrations”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3812–3822.
- [50] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).
- [51] Davis Meike and Leonids Ribickis. “Energy efficient use of robotics in the automobile industry”. In: *2011 15th international conference on advanced robotics (ICAR)*. IEEE. 2011, pp. 507–511.
- [52] Josh Merel et al. “Hierarchical visuomotor control of humanoids”. In: *International Conference in Learning Representations* (2019). URL: <http://arxiv.org/abs/1811.09656>.
- [53] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* (2015).



- [54] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
- [55] Jun Morimoto and Kenji Doya. “Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning”. In: *Robotics and Autonomous Systems* 36.1 (2001), pp. 37–51.
- [56] Ofir Nachum et al. “Data-efficient hierarchical reinforcement learning”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 3303–3313.
- [57] Anusha Nagabandi et al. “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7559–7566.
- [58] Ashvin V Nair et al. “Visual reinforcement learning with imagined goals”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9191–9200.
- [59] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.” In: *Icml*. Vol. 1. 2000, p. 2.
- [60] Shayegan Omidshafiei et al. “Deep decentralized multi-task multi-agent reinforcement learning under partial observability”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2681–2690.
- [61] OpenAI et al. “Learning Dexterous In-Hand Manipulation”. In: (2018), pp. 1–27.
- [62] Sinno Jialin Pan and Qiang Yang. “A survey on transfer learning”. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.
- [63] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. “Actor-mimic: Deep multitask and transfer reinforcement learning”. In: *arXiv preprint arXiv:1511.06342* (2015).
- [64] Xue Bin Peng et al. “MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies”. In: (May 2019). URL: <http://arxiv.org/abs/1905.09808>.
- [65] Jan Peters and Stefan Schaal. “Natural Actor-Critic”. In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190. ISSN: 09252312. DOI: [10.1016/j.neucom.2007.11.026](https://doi.org/10.1016/j.neucom.2007.11.026).
- [66] Lerrel Pinto and Abhinav Gupta. “Learning to push by grasping: Using multiple tasks for effective learning”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2161–2168.
- [67] Lerrel Pinto et al. “Asymmetric actor critic for image-based robot learning”. In: *arXiv preprint arXiv:1710.06542* (2017).
- [68] Matthias Plappert et al. “Multi-goal reinforcement learning: Challenging robotics environments and request for research”. In: *arXiv preprint arXiv:1802.09464* (2018).
- [69] Boris T Polyak and Anatoli B Juditsky. “Acceleration of stochastic approximation by averaging”. In: *SIAM Journal on Control and Optimization* (1992).

- [70] Doina Precup. *Temporal abstraction in reinforcement learning*. Jan. 2000. URL: <https://scholarworks.umass.edu/dissertations/AAI9978540>.
- [71] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. “Off-policy temporal-difference learning with function approximation”. In: *ICML*. 2001.
- [72] Pravesh Ranchod, Benjamin Rosman, and George Konidaris. “Nonparametric Bayesian Reward Segmentation for Skill Discovery Using Inverse Reinforcement Learning”. In: (2015). ISSN: 21530866. DOI: [10.1109/IR05.2015.7353414](https://doi.org/10.1109/IR05.2015.7353414).
- [73] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. “Maximum margin planning”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 729–736.
- [74] Andrei A Rusu et al. “Progressive neural networks”. In: *arXiv preprint arXiv:1606.04671* (2016).
- [75] Fereshteh Sadeghi and Sergey Levine. “(CAD)2RL: Real Single-Image Flight without a Single Real Image”. In: *arXiv preprint arXiv:1611.04201* (2016).
- [76] Kate Saenko et al. “Adapting visual category models to new domains”. In: *European conference on computer vision*. Springer. 2010, pp. 213–226.
- [77] Tom Schaul et al. “Universal value function approximators”. In: *ICML 2015*.
- [78] John Schulman et al. “HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION”. In: *International Conference in Learning Representations* (2016), pp. 1–14.
- [79] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: (2017). URL: <https://openai-public.s3-us-west-2.amazonaws.com/blog/2017-07/ppo/ppo-arxiv.pdf>.
- [80] John Schulman et al. “Trust Region Policy Optimization”. In: *International Conference in Machine Learning* (2015).
- [81] Archit Sharma et al. “Dynamics-aware unsupervised discovery of skills”. In: *arXiv preprint arXiv:1907.01657* (2019).
- [82] Arjun Sharma et al. “Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information”. In: *International Conference in Learning Representations* (2018). URL: <http://arxiv.org/abs/1810.01266>.
- [83] Tianmin Shu, Caiming Xiong, and Richard Socher. “Hierarchical and interpretable skill acquisition in multi-task reinforcement Learning”. In: *International Conference in Learning Representations 3* (2018), pp. 1–13. DOI: [10.1109/MWC.2016.7553036](https://doi.org/10.1109/MWC.2016.7553036).
- [84] David Silver et al. “Mastering the game of go without human knowledge”. In: *Nature* 550.7676 (2017), pp. 354–359.
- [85] David Silver et al. “Deterministic policy gradient algorithms”. In: *ICML 2014*.

- [86] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. ISSN: 14764687. DOI: [10.1038/nature24270](https://doi.org/10.1038/nature24270). URL: <http://arxiv.org/abs/1610.00633>.
- [87] Matthew J. A. Smith, Herke van Hoof, and Joelle Pineau. *An inference-based policy gradient method for learning options*. Feb. 2018. URL: <https://openreview.net/forum?id=rJIgf7bAZ>.
- [88] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. “Multitask Reinforcement Learning for Zero-shot Generalization with Subtask Dependencies”. In: *Advances in Neural Information Processing Systems* (2018).
- [89] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.
- [90] Richard S Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112 (1999), pp. 181–211. URL: <http://www-anw.cs.umass.edu/~barto/courses/cs687/Sutton-Precup-Singh-AIJ99.pdf>.
- [91] Richard S Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.
- [92] Yichuan Tang and Ruslan Salakhutdinov. “Learning Stochastic Feedforward Neural Networks”. In: *Advances in Neural Information Processing Systems 2* (2013), pp. 530–538. DOI: [10.1.1.63.1777](https://doi.org/10.1.1.63.1777).
- [93] Matthew E Taylor and Peter Stone. “Transfer learning for reinforcement learning domains: A survey”. In: *Journal of Machine Learning Research* 10.Jul (2009), pp. 1633–1685.
- [94] Josh Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *IROS* (2017).
- [95] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo : A physics engine for model-based control”. In: (2012), pp. 5026–5033.
- [96] Lisa Torrey and Jude Shavlik. “Transfer learning”. In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, 2010, pp. 242–264.
- [97] George Tucker et al. “The Mirage of Action-Dependent Baselines in Reinforcement Learning”. In: *International Conference in Machine Learning* (2018). URL: <http://arxiv.org/abs/1802.10031>.
- [98] Alexander Sasha Vezhnevets et al. “Feudal Networks for Hierarchical Reinforcement Learning”. In: *International Conference in Machine Learning* (2017). URL: <https://arxiv.org/pdf/1703.01161.pdf>.

- [99] Alexander Vezhnevets et al. “Strategic Attentive Writer for Learning Macro-Actions”. In: *Advances in Neural Information Processing Systems* (2016).
- [100] Théophane Weber et al. “Credit Assignment Techniques in Stochastic Computation Graphs”. In: (Jan. 2019). URL: <http://arxiv.org/abs/1901.01761>.
- [101] Ronald J Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 8.3-4 (1992), pp. 229–256.
- [102] Jun Yang, Rong Yan, and Alexander G Hauptmann. “Adapting SVM classifiers to data with shifted distributions”. In: *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE. 2007, pp. 69–76.
- [103] Tianhe Yu et al. “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning”. In: *arXiv preprint arXiv:1910.10897* (2019).
- [104] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning”. In: (2008).