Data-Efficient Hierarchical Reinforcement Learning

Ofir Nachum
Google Brain
ofirnachum@google.com

Shixiang Gu[*]
Google Brain
shanegu@google.com

Honglak Lee
Google Brain
honglak@google.com

Sergey Levine[†]
Google Brain
slevine@google.com

Abstract

Hierarchical reinforcement learning (HRL) is a promising approach to extend traditional reinforcement learning (RL) methods to solve more complex tasks. Yet, the majority of current HRL methods require careful task-specific design and on-policy training, making them difficult to apply in real-world scenarios. In this paper, we study how we can develop HRL algorithms that are general, in that they do not make onerous additional assumptions beyond standard RL algorithms, and efficient, in the sense that they can be used with modest numbers of interaction samples, making them suitable for real-world problems such as robotic control. For generality, we develop a scheme where lower-level controllers are supervised with goals that are learned and proposed automatically by the higher-level controllers. To address efficiency, we propose to use off-policy experience for both higher- and lower-level training. This poses a considerable challenge, since changes to the lower-level behaviors change the action space for the higher-level policy, and we introduce an off-policy correction to remedy this challenge. This allows us to take advantage of recent advances in off-policy model-free RL to learn both higher- and lower-level policies using substantially fewer environment interactions than on-policy algorithms. We term the resulting HRL agent *HIRO* and find that it is generally applicable and highly sample-efficient. Our experiments show that HIRO can be used to learn highly complex behaviors for simulated robots, such as pushing objects and utilizing them to reach target locations,[1] learning from only a few million samples, equivalent to a few days of real-time interaction. In comparisons with a number of prior HRL methods, we find that our approach substantially outperforms previous state-of-the-art techniques.[2]

1 Introduction

Deep reinforcement learning (RL) has made significant progress on a range of continuous control tasks, such as locomotion skills [39, 27, 18], learning dexterous manipulation behaviors [36], and training robot arms for simple manipulation tasks [13, 46]. However, most of these behaviors are inherently atomic: they require performing some simple skill, either episodically or cyclically, and rarely involve complex multi-level reasoning, such as utilizing a variety of locomotion behaviors to accomplish complex goals that require movement, object interaction, and discrete decision-making.

---

[*]Also at University of Cambridge; Max Planck Institute of Intelligent Systems.

[†]Also at UC Berkeley.

[1]See videos at https://sites.google.com/view/efficient-hrl

[2]Find open-source code at https://github.com/tensorflow/models/tree/master/research/efficient-hrl

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.
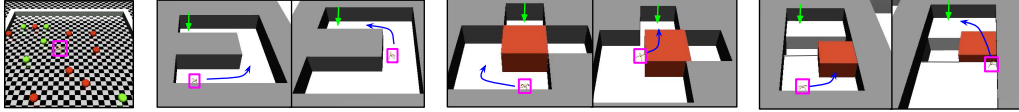
Figure 1: The Ant Gather task along with the three hierarchical navigation tasks we consider: Ant Maze, Ant Push, and Ant Fall. The ant (magenta rectangle) is rewarded for approaching the target location (green arrow). A successful policy must perform a complex sequence of directional movement and, in some cases, interact with objects in its environment (red blocks); e.g., pushing aside an obstacle (second from right) or using a block as a bridge (right). In our HRL method, a higher-level policy periodically produces goal states (corresponding to desired positions and orientations of the ant and its limbs), which the lower-level policy is rewarded to match (blue arrow).

Hierarchical reinforcement learning (HRL), in which multiple layers of policies are trained to perform decision-making and control at successively higher levels of temporal and behavioral abstraction, has long held the promise to learn such difficult tasks [7, 32, 43, 4]. By having a hierarchy of policies, of which only the lowest applies actions to the environment, one is able to train the higher levels to plan over a longer time scale. Moreover, if the high-level actions correspond to semantically different low-level behavior, standard exploration techniques may be applied to more appropriately explore a complex environment. Still, there is a large gap between the basic definition of HRL and the promise it holds to successfully solve complex environments. To achieve the benefits of HRL, there are a number of questions that one must suitably answer: How should one train the lower-level policy to induce semantically distinct behavior? How should the high-level policy actions be defined? How should the multiple policies be trained without incurring an inordinate amount of experience collection? Previous work has attempted to answer these questions in a variety of ways and has provided encouraging successes [48, 10, 11, 19, 40]. However, many of these methods lack generality, requiring some degree of manual task-specific design, and often require expensive on-policy training that is unable to benefit from advances in off-policy model-free RL, which in recent years has drastically brought down sample complexity requirements [12, 16, 3].

For generality, we propose to take advantage of the state observation provided by the environment to the agent, which in locomotion tasks can include the position and orientation of the agent and its limbs. We let the high-level actions be goal states and reward the lower-level policy for performing actions which yield it an observation close to matching the desired goal. In this way, our HRL setup does not require a manual or multi-task design and is fully general.

This idea of a higher-level policy commanding a lower-level policy to match observations to a goal state has been proposed before [7, 48]. Unlike previous work, which represented goals and rewarded matching observations within a learned embedding space, we use the state observations in their raw form. This significantly simplifies the learning, and in our experiments, we observe substantial benefits for this simpler approach.

While these goal-proposing methods are very general, they require training with on-policy RL algorithms, which are generally less efficient than off-policy methods [15, 31]. On-policy training has been attractive in the past since, outside of discrete control, off-policy methods have been plagued with instability [15], which is amplified when training multiple policies jointly, as in HRL. Other than instability, off-policy training poses another challenge that is unique to HRL. Since the lower-level policy is changing underneath the higher-level policy, a sample observed for a certain high-level action in the past may not yield the same low-level behavior in the future, and thus not be a valid experience for training. This amounts to a non-stationary problem for the higher-level policy. We remedy this issue by introducing an off-policy correction, which re-labels an experience in the past with a high-level action chosen to maximize the probability of the past lower-level actions. In this way, we are able to use past experience for training the higher-level policy, taking advantage of progress made in recent years to provide stable, robust, and general off-policy RL methods [12, 31, 3].

In summary, we introduce a method to train a multi-level HRL agent that stands out from previous methods by being both generally applicable and data-efficient. Our method achieves generality by training the lower-level policy to reach goal states learned and instructed by the higher-levels. In contrast to prior work that operates in this goal-setting model, we use states as goals directly, which allows for simple and fast training of the lower layer. Moreover, by using off-policy training with

2

our novel off-policy correction, our method is extremely sample-efficient. We evaluate our method on several difficult environments. These environments require the ability to perform exploratory navigation as well as complex sequences of interaction with objects in the environment (see Figure 1). While these tasks are unsolvable by existing non-HRL methods, we find that our HRL setup can learn successful policies. When compared to other published HRL methods, we also observe the superiority of our method, in terms of both final performance and speed of learning. In only a few million experience samples, our agents are able to adequately solve previously unapproachable tasks.

2 Background

We adopt the standard continuous control RL setting, in which an agent interacts with an environment over periods of time according to a behavior policy μ. At each time step t, the environment produces a state observation $s_t \in \mathbb{R}^{d_s}$. The agent then samples an action $a_t \sim \mu(s_t), a_t \in \mathbb{R}^{d_a}$ and applies the action to the environment. The environment then yields a reward R_t sampled from an unknown reward function $R(s_t, a_t)$ and either terminates the episode at state s_T or transitions to a new state s_{t+1} sampled from an unknown transition function $f(s_t, a_t)$. The agent's goal is to maximize the expected future discounted reward $\mathbb{E}_{s_{0:T}, a_{0:T-1}, R_{0:T-1}} \left[\sum_{i=0}^{T-1} \gamma^i R_i \right]$, where $0 \leq \gamma < 1$ is a user-specified discount factor. A well-performing RL algorithm will learn a good behavior policy μ from (ideally a small number of) interactions with the environment.

2.1 Off-Policy Temporal Difference Learning

Temporal difference learning is a powerful paradigm in RL, in which a policy may be learned efficiently from state-action-reward transition tuples (s_t, a_t, R_t, s_{t+1}) collected from interactions with the environment. In our HRL method, we utilize the TD3 learning algorithm [12], a variant of the popular DDPG algorithm for continuous control [27].

In DDPG, a deterministic neural network policy μ_ϕ is learned along with its corresponding state-action Q-function Q_θ by performing gradient updates on parameter sets ϕ and θ. The Q-function represents the future value of taking a specific action a_t starting from a state s_t. Accordingly, it is trained to minimize the average Bellman error over all sampled transitions, which is given by

$$\mathcal{E}(s_t, a_t, s_{t+1}) = (Q_\theta(s_t, a_t) - R_t - \gamma Q_\theta(s_{t+1}, \mu_\phi(s_{t+1})))^2. \tag{1}$$

The policy is then trained to yield actions which maximize the Q-value at each state. That is, μ_ϕ is trained to maximize $Q_\theta(s_t, \mu_\phi(s_t))$ over all s_t collected from interactions with the environment.

We note that although DDPG trains a deterministic policy μ_ϕ, its behavior policy, which is used to collect experience during training is augmented with Gaussian (or Ornstein-Uhlenbeck) noise [27]. Therefore, actions are collected as $a_t \sim N(\mu_\phi(s_t), \sigma)$ for fixed standard deviation σ, which we will shorten as $a_t \sim \mu_\phi(s_t)$. We will take advantage of the fact that the behavior policy is stochastic for the off-policy correction in our HRL method. TD3 [12] makes several modifications to DDPG's learning algorithm to yield a more robust and stable procedure. Its main modification is using an ensemble over Q-value models and adding noise to the policy when computing the target value in Equation 1.

3 General and Efficient Hierarchical Reinforcement Learning

In this section, we present our framework for learning hierarchical policies, HIRO: HIerarchical Reinforcement learning with Off-policy correction. We make use of parameterized reward functions to specify a potentially infinite set of lower-level policies, each of which is trained to match its observed states s_t to a desired goal. The higher-level policy chooses these goals for temporally extended periods, and uses an off-policy correction to enable it to use past experience collected from previous, different instantiations of the lower-level policy.

3.1 Hierarchy of Two Policies

We extend the standard RL setup to a hierarchical two-layer structure, with a lower-level policy μ^{lo} and a higher-level policy μ^{hi} (see Figure 2). The higher-level policy operates at a coarser layer

3

1. Collect experience $s_t, g_t, a_t, R_t, \ldots$.

2. Train μ^{lo} with experience transitions $(s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1})$ using g_t as additional state observation and reward given by goal-conditioned function $r_t = r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||_2$.

3. Train μ^{hi} on temporally-extended experience $(s_t, \tilde{g}_t, \sum R_{t:t+c-1}, s_{t+c})$, where $\tilde{g}_t$ is re-labelled high-level action to maximize probability of past low-level actions $a_{t:t+c-1}$.

4. Repeat.

Figure 2: The design and basic training of HIRO. The lower-level policy interacts directly with the environment. The higher-level policy instructs the lower-level policy via high-level actions, or goals, $g_t \in \mathbb{R}^{d_s}$ which it samples anew every c steps. On intermediate steps, a fixed goal transition function h determines the next step's goal. The goal simply instructs the lower-level policy to reach specific states, which allows the lower-level policy to easily learn from prior off-policy experience.
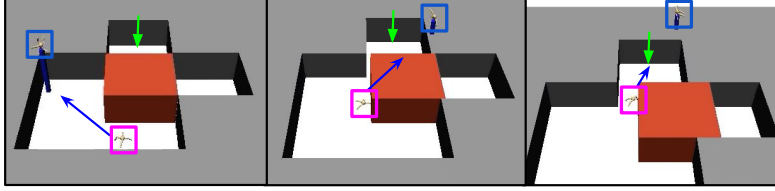
Figure 3: An example of a higher-level policy producing goals in terms of desired observations, which in this task correspond to positions and orientations of all of the joints of a quadrupedal robot (including root position). The lower-level policy has direct control of the agent (pink), and is rewarded for matching the position and orientation of its torso and each limb to the goal (blue rectangle, raised for visibility). In this way, the two-layer policy can perform a complex task involving a sequence of movements and interactions; e.g. pushing a block aside to reach a target (green).

of abstraction and sets goals to the lower-level policy, which correspond directly to states that the lower-level policy attempts to reach. At each time step t, the environment provides an observation state s_t. The higher-level policy observes the state and produces a *high-level action* (or *goal*) $g_t \in \mathbb{R}^{d_s}$ by either sampling from its policy $g_t \sim \mu^{hi}$ when $t \equiv 0 \pmod{c}$, or otherwise using a fixed goal transition function $g_t = h(s_{t-1}, g_{t-1}, s_t)$ (which in the simplest case can be a pass-through function, although we will consider a slight variation in our specific design). This provides temporal abstraction, since high-level decisions via μ^{hi} are made only every c steps. The lower-level policy μ^{lo} observes the state s_t and goal g_t and produces a low-level atomic action $a_t \sim \mu^{lo}(s_t, g_t)$, which is applied to the environment. The environment then yields a reward R_t sampled from an unknown reward function $R(s_t, a_t)$ and transitions to a new state s_{t+1} sampled from an unknown transition function $f(s_t, a_t)$.

The higher-level controller provides the lower-level with an intrinsic reward $r_t = r(s_t, g_t, a_t, s_{t+1})$, using a fixed parameterized reward function r. The lower-level policy will store the experience $(s_t, g_t, a_t, r_t, s_{t+1}, h(s_t, g_t, s_{t+1}))$ for off-policy training. The higher-level policy collects the environment rewards R_t and, every c time steps, stores the higher-level transition $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$ for off-policy training.

3.2 Parameterized Rewards

Our higher-level policy produces goals g_t indicating desired relative changes in state observations. That is, at step t, the higher-level policy produces a goal g_t, indicating its desire for the lower-level agent to take actions that yield it an observation s_{t+c} that is close to $s_t + g_t$. Although some state dimensions (e.g., the position of the quadrupedal robot in Figure 3) are more natural as goal subspaces, we chose this more generic goal representation to make it broadly applicable, without any manual design of goal spaces, primitives, or controllable dimensions. This makes our method general and

4

easy to apply to new problem settings. To maintain the same absolute position of the goal regardless of state change, the goal transition model h is defined as

$$h(s_t, g_t, s_{t+1}) = s_t + g_t - s_{t+1}. \tag{2}$$

We define the intrinsic reward as a parameterized reward function based on the distance between the current observation and the goal observation:

$$r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||_2. \tag{3}$$

This rewards the lower-level policy for taking actions that yield observations that are close to the desired value $s_t + g_t$. In our evaluations on simulated ant locomotion, we use all positional observations as the representation for g_t, without distinguishing between the (x, y, z) root position or the joints, making for a generic and broadly applicable choice of goal space. The reward r and transition function h are computed only with respect to these positional observations. See Figure 3 for an example of the goals g_t chosen during a successful navigation of a complex environment.

The lower-level policy may be trained using standard methods by simply incorporating g_t as an additional input into the value and policy models. For example, in DDPG, the equivalent objective to Equation 1 in terms of lower-level Q-value function Q_θ^{lo} is to minimize the error

$$(Q_\theta^{lo}(s_t, g_t, a_t) - r(s_t, g_t, a_t, s_{t+1}) - \gamma Q_\theta^{lo}(s_{t+1}, g_{t+1}, \mu_\phi^{lo}(s_{t+1}, g_{t+1})))^2, \tag{4}$$

for all transitions $(s_t, g_t, a_t, s_{t+1}, g_{t+1})$. The policy μ_ϕ^{lo} would be trained to maximize the Q-value $Q_\theta^{lo}(s_t, g_t, \mu_\phi^{lo}(s_t, g_t))$ for all sampled state-goal tuples (s_t, g_t).

Parameterized rewards are not a new concept, and have been studied previously [38, 20]. They are a natural choice for a generally applicable HRL method and have therefore appeared as components of other HRL methods [48, 24, 33, 26]. A significant distinction between our method and these prior approaches is that we directly use the state observation as the goal, and changes in the state observation as the action space for the higher-level policy, in contrast to prior methods that must train the goal representation. This allows the lower-level policy to begin receiving reward signals immediately, even before the lower-level policy has figured out how to reach the goal and before the task's extrinsic reward provides any meaningful supervision. In our experiments (Section 5), we find that this produces substantially better results.

3.3 Off-Policy Corrections for Higher-Level Training

While a number of prior works have proposed two-level HRL architectures that involve some sort of goal setting, such designs in previous work generally require on-policy training [48]. This is because the changing behavior of the lower-level policy creates a non-stationary problem for the higher-level policy, and old off-policy experience may exhibit different transitions conditioned on the same goals. However, for HRL methods to be applicable to real-world settings, they must be sample-efficient, and off-policy algorithms (often based on some variant of Q-function learning) generally exhibit substantially better sample efficiency than on-policy actor-critic or policy gradient variants. In this section, we describe how we address the challenge of off-policy training of the higher-level policy.

We would like to take the higher-level transition tuples $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$, where $x_{t:t+c-1}$ denotes the sequence $x_t, \ldots, x_{t+c-1}$, which are collected by the higher-level policy and convert them to state-action-reward transitions $(s_t, g_t, \sum R_{t:t+c-1}, s_{t+c})$ that can be pushed into the replay buffer of any standard off-policy RL algorithm. However, since transitions obtained from past lower-level controllers do not accurately reflect the actions (and therefore resultant states $s_{t+1:t+c}$) that would occur if the same goal were used with the current lower-level controller, we must introduce a correction that translates old transitions into ones that agree with the current lower-level controller.

Our main observation is that the goal g_t of a past high-level transition $(s_t, g_t, \sum R_{t:t+c-1}, s_{t+c})$ may be changed to make the actual observed action sequence more likely to have happened with respect to the current instantiation of μ^{lo}. The high-level action g_t which in the past induced a low-level behavior $a_{t:t+c-1} \sim \mu^{lo}(s_{t:t+c-1}, g_{t:t+c-1})$ may be re-labeled to a goal $\tilde{g}_t$ which is likely to induce the same low-level behavior with the current instantiation of the lower-level policy. Thus, we propose to remedy the off-policy issue by re-labeling the high-level transition $(s_t, g_t, \sum R_{t:t+c-1}, s_{t+c})$ with a different high-level action $\tilde{g}_t$ chosen to maximize the probability $\mu^{lo}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$,

5

where the intermediate goals $\tilde{g}_{t+1:t+c-1}$ are computed using the fixed goal transition function h. In effect, each time we modify the low-level policy μ^{lo}, we would like to answer the question: for which goals would this new controller have taken the same actions as the old one?

Most RL algorithms will use random action-space exploration to select actions, which means that the behavior policy (even for deterministic algorithms such as DDPG [27]) is stochastic and the log probability $\log \mu^{lo}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$ may be computed as

$$\log \mu^{lo}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1}) \propto -\frac{1}{2} \sum_{i=t}^{t+c-1} ||a_i - \mu^{lo}(s_i, \tilde{g}_i)||_2^2 + \text{const.} \qquad (5)$$

To approximately maximize this quantity in practice, we compute this log probability for a number of goals $\tilde{g}_t$, and choose the maximal goal to re-label the experience. In our implementation, we calculate the quantity on eight candidate goals sampled randomly from a Gaussian centered at $s_{t+c} - s_t$. We also include the original goal g_t and a goal corresponding to the difference $s_{t+c} - s_t$ in the candidate set, to have a total of 10 candidates. This provides a suitably diverse set of $\tilde{g}_t$ to approximately solve the $\arg\max$ of Equation 5, while also biasing the result to be closer to candidates $\tilde{g}_t$ which we believe to be appropriate given our knowledge of the problem (see additional implementation details in the Appendix). Our approach here is only an approximation, and we elaborate on possible alternative off-policy corrections in the Appendix.

4 Related Work

Discovering meaningful and effective hierarchies of policies is a long standing research problem in RL [7, 32, 43, 8, 2]. Classically, the work on HRL focused on discrete state domains, where state visitation and transition statistics can be used to construct heuristic sub-goals for low-level policies [41, 29, 5]. The options framework [43, 35], a popular formulation for HRL, proposes a termination policy for each sub-policy (option). While the traditional options framework relies on prior knowledge for designing options, [2] recently derived an actor-critic algorithm for learning them jointly with the higher-level policy. This option-critic architecture [2] is an important step toward end-to-end HRL; however, such approaches are often prone to learning either a sub-policy that terminates every time step, or one effective sub-policy that runs through the whole episode. In practice, regularizers are essential to learn multiple effective and temporally abstracted sub-policies [2, 17, 47].

To guarantee learning useful sub-policies, recent work has studied approaches that provide auxiliary rewards for the low-level policies [5, 19, 24, 44, 10]. These approaches rely on hand-crafted rewards based on prior domain knowledge [23, 19, 24, 44] or diversity-encouraging rewards like mutual information [6, 10]. A number of works have suggested that semantically distinct behavior can be induced by training on a set of diverse tasks, and have suggested pre-training the lower-level policy on such tasks [19, 10], or training the multi-level hierarchical policy in a multi-task setup [11, 40]. However, having access to a collection of suitably similar tasks is a luxury which is not always available and may require hand-design. Our method uses a generic reward that is specified with respect to the state space, and therefore avoids designing various rewards or multiple tasks.

Another difference from most HRL work [10, 11] is that we use off-policy learning, leading to significant improvements in sample efficiency. In end-to-end HRL, off-policy RL creates a non-stationary problem for the higher-level policy, since the lower-level is constantly changing. We are aware of only one recent work which applies HRL in an off-policy setting [26]. As in our work, the authors devise a hierarchical structure in which a lower-level policy is trained to reach observations directed by a higher-level policy. The multiple layers of policies are trained jointly in an off-policy manner, while ignoring the non-stationarity problem which we realize is a key issue for off-policy HRL. Accordingly, we derive and test an off-policy correction in the context of HRL, and empirically show that this technique is crucial to successfully train hierarchical policies on complex tasks.

Our work is related to FeUdal Networks (FuN) [48], originally inspired from feudal RL [7]. FuN also makes use of goals and a parameterized lower-level reward. Unlike our method, FuN represents the goals and computes the rewards in terms of a learned state representation. In our experiments, we found this technique to under-perform compared to our approach, which uses the state in its raw form. We find that this has a number of benefits. For one, the lower-level policies can immediately begin receiving intrinsic rewards for reaching goals even before the higher-level policy receives a meaningful supervision signal from the task reward. Additionally, the representation is generic and

6

	Ant Gather	Ant Maze	Ant Push	Ant Fall
HIRO	**3.02±1.49**	**0.99±0.01**	**0.92±0.04**	**0.66±0.07**
FuN representation	0.03 ± 0.01	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
FuN transition PG	0.41 ± 0.06	0.0 ± 0.0	0.56 ± 0.39	0.01 ± 0.02
FuN cos similarity	0.85 ± 1.17	0.16 ± 0.33	0.06 ± 0.17	0.07 ± 0.22
FuN	0.01 ± 0.01	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
SNN4HRL	1.92 ± 0.52	0.0 ± 0.0	0.02 ± 0.01	0.0 ± 0.0
VIME	1.42 ± 0.90	0.0 ± 0.0	0.02 ± 0.02	0.0 ± 0.0

Table 1: Performance of the best policy obtained in 10M steps of training, averaged over 10 randomly seeded trials with standard error. Comparisons are to variants of FuN [48], SNN4HRL [10], and VIME [21]. Even after extensive hyper-parameter searches, we were unable to achieve competitive performance from the baselines on any of our tasks. In the Appendix, we include the only competitive result we could achieve – VIME on Ant Gather trained for a much longer amount of time.

simple to obtain. Goal-conditioned value functions [28, 42, 38, 1, 34] are actively explored outside the context of HRL. Continued progress in this field may be used to further improve HRL methods.

5 Experiments

In our experiments, we compare HIRO method to prior techniques, and ablate the various components to understand their importance. Our experiments are conducted on a set of challenging environments that require a combination of locomotion and object manipulation. Visualizations of these environments are shown in Figure 1. See the Appendix for more details on each environment.

Ant Gather. The ant gather task is a standard task introduced in [9]. A simulated ant must navigate to gather apples while avoiding bombs, which are randomly placed in the environment at the beginning of each episode. The ant receives a reward of 1 for each apple and a reward of -1 for each bomb.

Ant Maze. For the first difficult navigation task we adapted the maze environment introduced in [9]. In this environment an ant must navigate to various locations in a '⊃'-shaped corridor. We increase the default size of the maze so that the corridor is of width 8. In our evaluation, we assess the success rate of the policy when attempting to reach the end of the maze.

Ant Push. In this task we introduce a movable block which the agent can interact with. A greedy agent would move forward, unknowingly pushing the movable block until it blocks its path to the target. To successfully reach the target, the ant must first move to the left around the block and then push the block right, clearing the path towards the target location.

Ant Fall. This task extends the navigation to three dimensions. The ant is placed on a raised platform, with the target location directly in front of it but separated by a chasm which it cannot traverse by itself. Luckily, a movable block is provided on its right. To successfully reach the target, the ant must first walk to the right, push the block into the chasm, and then safely cross.

5.1 Comparative Analysis

The primary comparisons to previous HRL methods are done with respect to FeUdal Networks (FuN) [48], stochastic neural networks for HRL (SNN4HRL) [10], and VIME [21] (see Table 1, and Appendix for more details). As these algorithms often come with problem-specific design choices, we modify each for fairer comparisons. In terms of problem assumptions, our work is closest to that of FuN which is applicable to any single task without specific sub-policy reward engineering. MLSH [11] is another promising recent work for HRL; however, since it relies on learning meaningful sub-policies through experiencing multiple, diverse, hand-designed tasks, we do not include explicit comparisons. We leave exploring our method in the context of multi-task learning for future work.

FeUdal Network (FuN). Unlike SNN4HRL or VIME, the official open-source code for FuN was not available at the time of submission, and therefore we aimed to replicate key design choices of FuN from our algorithm implementation. FuN [48] primarily proposes four components: (1) transition policy gradient, (2) directional cosine similarity rewards, (3) goals specified with respect to a learned representation, and (4) dilated RNN. Since our tasks are low-dimensional and fully observed, we do

7
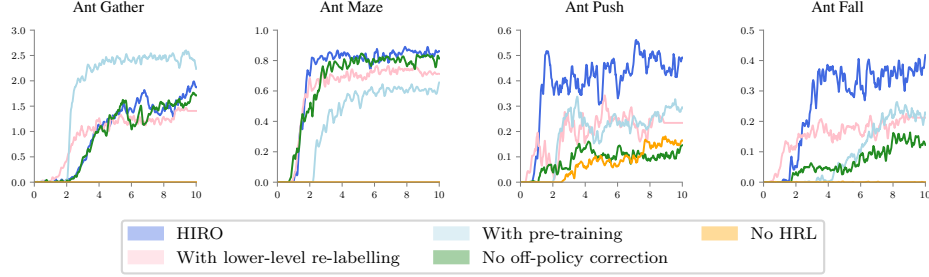
Figure 4: Results of our method and a number of variants on a set of difficult tasks. Each plot shows average reward (for Ant Gather) or average success rate (for the rest; see Appendix) over 10 randomly seeded trials, with x-axis in millions of environment steps. We find that HIRO can perform well across all tasks. We also note that HIRO learns rapidly; on the complex navigation tasks it requires only a few million environment steps (a few days in real-world interaction time) to achieve good performance. Our method is only out-performed on Ant Gather by a variant that pre-trains the lower-level policy (thus not needing an off-policy correction).

not include design choice (4). For each of (1), (2), and (3), we apply an equivalent modification of our HRL method and evaluate its performance on the same tasks. We also evaluate all modifications together as an approximation to the entire FuN paradigm. Results in Table 1 show that on our tasks, the FuN modifications do not learn well, and other than Ant Gather are significantly out-performed by HIRO. In particular, it is worth noting that the use of learned representations, rather than observation goals, leads to almost no improvement on the tasks. This suggests that the choice of using goal observations as lower-level goals significantly improves HRL performance, by providing a strong supervision signal to the lower-level policy right from the beginning of training.

Stochastic Neural Networks for HRL (SNN4HRL). SNN4HRL [10] initially trains the low-level policy with a proxy reward to encourage learning useful diverse exploration policies, and then the high-level policy is trained in the tasks of interest while the low-level is fixed. While SNN4HRL can perform better than FuN, it is still far behind our proposed HRL method.

Variational Information Maximizing Exploration (VIME). VIME [21] is not an HRL method but is used as a strong baseline in SNN4HRL. As discussed in [10] and matched by our results, for the benchmark's short horizon task of length 500, it performs approximately the same as SNN4HRL.

Option-Critic Architecture. We extended the option-critic architecture implementation [2] for continuous actions and attempted a number of alternative variants besides the naïve modification of the original. No versions yielded reasonable performance in our tasks, and so we omit it from the results. This is possibly due to difficulty in continuous control tasks, but most importantly the option-critic sub-policies rely solely on the external reward, making learning gait policies difficult.

5.2 Ablative Analysis

In Figure 4 we present results of our proposed HRL method ("HIRO") compared with a number of variants to understand the importance of various design choices:

With lower-level re-labelling. We evaluate the benefit of recent proposals [1, 25] to increase the amount of data available to an agent trained using a parameterized reward (the lower-level policy in our setup) by re-labeling experiences with randomly sampled goals. This allows the lower-level policy to use experience collected with respect to a specific goal g to be used to learn behavior with respect to any alternative goal $\tilde{g}$. Our results show that this technique can provide an initial speed-up in training; however, its performance is quick to plateau. We hypothesize that re-labeling goals randomly may make lower-level training more difficult, since the policy must learn to not only satisfy the goals provided by the higher-level agent, but instead almost any conceivable goal. The benefit of re-labeling goals will require more research, and we encourage future work to investigate better ways to harness its benefits.

8

With pre-training. In this variant we evaluate a simpler method to avoid the non-stationary issue in higher-level off-policy training. Rather than correct for past experiences, we instead pre-train the lower-level policy for 2M steps (using goals sampled from a Gaussian) before freezing it and training the higher-level policy alone (this variant also has the advantage of allowing the higher-level policy to learn with respect to a deterministic, non-exploratory lower-level policy). In the harder navigation tasks, we find that pre-training is detrimental. This is understandable, as these tasks require specialization in different low-level behavior for different stages of the navigation. By allowing the lower-level policy to continually learn as new parts of the environment are encountered, we are able to learn a lower-level policy which is better able to satisfy the desired goals of the higher-level. In contrast, in the simpler and mostly homogeneous Ant Gather task, the advantage of pre-training is significant. This suggests that our off-policy correction is still not perfect, and there is potentially significant benefit to be obtained by improving it.

No off-policy correction. We assess the advantage of including the off-policy correction compared to training off-policy naïvely, ignoring the non-stationary issue. Interestingly, training an HRL policy this way can do quite well. However, in the harder tasks (Ant Push, Ant Fall) the issue becomes difficult to ignore. Accordingly, we observe a significant benefit from using the off-policy correction.

No HRL. Finally, we evaluate the ability of a single non-HRL policy to learn in these environments. This variant makes almost no progress on the tasks compared to our HRL method.

6 Conclusion

We have presented a method for training a two-layer hierarchical policy. Our approach is general, using learned goals to pass instructions from the higher-level policy to the lower-level one. Moreover, we have described a method by which both polices may be trained in an off-policy manner concurrently for highly sample-efficient learning. Our experiments show that our method outperforms prior HRL algorithms and can solve exceedingly complex tasks that combine locomotion and rudimentary object interaction. We note that our results are still far from perfect, and there is much work left for future research to improve the stability and performance of HRL methods on these tasks.

7 Acknowledgments

We thank Ben Eysenbach and others on the Google Brain team for insightful comments and discussions.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

[3] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[4] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[5] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

[6] Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281, 2012.

[7] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

9

[8] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[9] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[10] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[11] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *International Conference on Learning Representations (ICLR)*, 2018.

[12] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[13] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[14] Shixiang Gu, Tim Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3849–3858, 2017.

[15] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

[16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[17] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. *arXiv preprint arXiv:1709.04571*, 2017.

[18] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[19] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[20] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.

[21] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[24] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[25] Sergey Levine, Shane Gu, and Vitchyr Pong. Temporal difference model learning: Model-free deep rl for model-based control. 2018.

[26] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical actor-critic. *arXiv preprint arXiv:1712.00948*, 2017.

[27] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

10

[28] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231, 2007.

[29] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.

[30] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[31] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Trust-pcl: An off-policy trust region method for continuous control. *arXiv preprint arXiv:1707.01891*, 2017.

[32] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.

[33] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[34] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *International Conference on Learning Representations*, 2018.

[35] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[36] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[37] Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6553–6564, 2017.

[38] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[39] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[40] Olivier Sigaud and Freek Stulp. Policy search in continuous action domains: an overview. *arXiv preprint arXiv:1803.04706*, 2018.

[41] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

[42] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[43] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[44] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, page 6, 2017.

[45] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[46] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

11

[47] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pages 3486–3494, 2016.

[48] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[49] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *International Conference on Learning Representations*, 2017.

A Discussion on Alternative Off-Policy Corrections for High-Level Actions

Through our experiments, we found that our proposed maximum likelihood-based action relabeling works well empirically; however, we also tried other variants of off-policy correction schemes. While none of the methods below worked as well as ours in the tested domains based on preliminary experiments, we summarize them below as a reference for further future work on off-policy correction for HRL.

The experience replay stores $(s_{t:t+c}, a_{t:t+c-1}, g_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$ sampled from following a low-level policy $a_i \sim \mu_\beta^{lo}(a_i|s_i, g_i)$. a_i is low-level action and g_i is high-level action (or goal for the low-level policy). We want to estimate the following objective for the current low-level policy $\mu^{lo}(a|s, g)$, where Q^{hi} represents the target network,

$$L(\theta) = \mathbb{E}_\beta \left[\left(Q_\theta^{hi}(s_t, g_t) - y_t \right)^2 \right] \tag{6}$$

$$y_t = \mathbb{E}_{\prod_{i=t}^{t+c-1} \mu^{lo}(a_i|s_i, g_i) p(s_{i+1}|s_t, a_i)} \left[R_{t:t+c-1} + \gamma \max_g Q^{hi}(s_{t+c}, g) \right] \tag{7}$$

$$= \mathbb{E}_{\prod_{i=t}^{t+c-1} \mu_\beta^{lo}(a_i|s_i, g_i) p(s_{i+1}|s_t, a_i)} \left[w_t \cdot \left(R_{t:t+c-1} + \gamma \max_g Q^{hi}(s_{t+c}, g) \right) \right] \tag{8}$$

$$w_t = \prod_{i=t}^{t+c-1} \frac{\mu^{lo}(a_i|s_i, g_i)}{\mu_\beta^{lo}(a_i|s_i, g_i)}. \tag{9}$$

We remind the reader that g_i is computed using a deterministic dynamics from g_t using $g_{i+1} = h(s_t, g_t, s_{t+1}) = s_i + g_i - s_{i+1}$ for $i = t, t+1, \ldots, t+c-2$.

Direct Importance Correction. A naïve approach is to directly use the unbiased estimator based on importance weighting defined by the expectation in Eq. 9,

$$L(\theta) = \mathbb{E}_\beta \left[\left(Q_\theta^{hi}(s_t, g_t) - \hat{y}_t \right)^2 \right] \tag{10}$$

$$\hat{y}_t = w_t \left(R_{t:t+c-1} + \gamma \max_g Q^{hi}(s_{t+c}, g) \right) \tag{11}$$

$$w_t = \prod_{i=t}^{t+c-1} \frac{\mu^{lo}(a_i|s_i, g_i)}{\mu_\beta^{lo}(a_i|s_i, g_i)}. \tag{12}$$

For the continuous action domains in our paper, we found this estimator, while unbiased, has very high variance, and does not work well in practice.

Importance-Based Action Relabeling. Instead of computing the high-variance importance weight for the sample goal g_t, we may also try to find a new goal $\tilde{g}_t$ such that the importance weight is approximately 1. This leads to the action relabeling objective as used in our method,

$$L(\theta) = \mathbb{E}_\beta \left[\left(Q_\theta^{hi}(s_t, \tilde{g}_t) - \hat{y}_t \right)^2 \right] \tag{13}$$

$$\hat{y}_t = R_{t:t+c-1} + \gamma \max_g Q^{hi}(s_{t+c}, g), \tag{14}$$

12

where $\tilde{g}_t$ can be found by minimizing loss functions such as,

$$\tilde{g}_t = \arg\min_{g_t} \left(1 - \prod_{i=t}^{t+c-1} \frac{\mu^{lo}(a_i|s_i,g_i)}{\mu_\beta^{lo}(a_i|s_i,g_i)} \right)^2 \tag{15}$$

$$\tilde{g}_t = \arg\min_{g_t} \left(\sum_{i=t}^{t+c-1} \log \mu^{lo}(a_i|s_i,g_i) - \log \mu_\beta^{lo}(a_i|s_i,g_i) \right)^2. \tag{16}$$

Since there is no guarantee that $\tilde{g}_t$ exists to make the loss function go to 0, this estimator is still biased. However, we could expect that the bias may be reduced.

Model-Based Relabeling. What we need to ensure for off-policy correction is that $(s_{t:t+c-1}, g_{t:t+c-1}, s_{t+c})$ is consistent with the dynamics of MDP transition $p(s_{i+1}|s_i,a_i)$ and current low-level policy $\mu^{lo}(a_i|s_i,g_i)$. If we can approximate either the high-level forward dynamics $\tilde{s}_{t+c} = p^{hi}(\cdot|s_t,g_t)$ or the inverse model $\tilde{g}_t \sim p_{inv}^{hi}(\cdot|s_t,s_{i+c})$, then we may directly do model-based prediction to relabel for either s_{t+c} or g_t. While the action relabeling TD objective is given as Eq. 14, the state relabeling objective is given by,

$$L(\theta) = \mathbb{E}_\beta \left[\left(Q_\theta^{hi}(s_t,g_t) - \hat{y}_t \right)^2 \right] \tag{17}$$

$$\hat{y}_t = R_{t:t+c-1} + \gamma \max_g Q^{hi}(\tilde{s}_{t+c},g). \tag{18}$$

The question is how to get p^{hi} or p_{inv}^{hi}. While we can fit parametric functions on samples of data, this is often as difficult as fully model-based approach. We may instead make use of that fact that the low-level is trying to reach the given goal states. Assuming the low-level policy eventually gets to complete the given goals, we may use the following forms,

$$p^{hi}(\tilde{s}_{t+c}|s_t,g_t) = \mathcal{N}(s_t + g_t, \Sigma) \tag{19}$$

$$p_{inv}^{hi}(\tilde{g}_t|s_t,s_{t+c}) = \mathcal{N}(s_{t+1} - s_t, \Sigma). \tag{20}$$

This resembles transition policy gradient in FuN [48], where the high-level policy is trained by assuming the low-level approximately completes the assigned goals. Empirically, we did not observe this outperformed our approach on the tested domains.

B   Environment Details

Environments use the MuJoCo simulator [45] with $dt = 0.02$ and frame skip set to 5.

B.1   Gather

We use the Gather environment provided by Rllab with a simulated ant agent. The ant is equivalent to the standard Rllab Ant, except that its gear range is reduced from $(-150, 150)$ to $(-30, 30)$. In addition to observing $qpos$, $qvel$, and the current time step t, the agent also observes depth readings as defined by the standard Gather environment. We set the activity range to 10 and the sensor span to 2π, which matches the settings in [10].

Each episode is terminated either when the ant falls or at 500 steps.

The reward used is the default reward (number of apples minus number of bombs).

B.2   Navigation

We devise three navigation tasks to evaluate our method. In each navigation task, we create an environment of $8 \times 8 \times 8$ blocks, some movable and some with fixed position. We use the same ant agent used in Gather. The agent observes $qpos$, $qvel$, the current time step t, and the target location. Its actions correspond to torques applied to joints. At the beginning of each episode, the environment samples a target position (g_x, g_y) and the agent is provided a reward at each step corresponding to negative L2 distance from the target: $-\sqrt{(g_x - x)^2 + (g_y - y)^2}$. In one of the navigation tasks (Falling), the L2 distance is measured with respect to 3 coordinates: x, y, and z. Each episode is 500 steps long (i.e., the episode does not terminate when the ant falls).

We describe the specifics of each navigation task below.

13

B.2.1 Maze

In this task, immovable blocks are placed to confine the agent to a "⊃"-shaped corridor. That is, blocks are placed everywhere except at $(0,0), (8,0), (16,0), (16,8), (16,16), (8,16), (0,16)$. The agent is initialized at position $(0,0)$. At each episode, a target position is sampled uniformly at random from $g_x \sim [-4, 20], g_y \sim [-4, 20]$.

At evaluation time, we evaluate the agent only on its ability to reach $(0, 16)$. We define a "success" as being within an L2 distance of 5 from the target on the ultimate step of the episode.

B.2.2 Push

In this task, immovable blocks are placed everywhere except at $(0,0), (-8,0), (-8,8), (0,8), (8,8), (16,8), (0,16)$. A movable block is placed at $(0,8)$. The agent is initialized at position $(0,0)$. At each episode, the target position is fixed to $(g_x, g_y) = (0, 19)$. Therefore, the agent must first move to the left, the push the movable block to the right, and then navigate to the target unimpeded.

At evaluation time, we evaluate the agent on its ability to reach $(0, 19)$. We define a "success" as being within an L2 distance of 5 from the target on the ultimate step of the episode.

B.2.3 Fall

In this task, the agent is initialized on a platform of height 4. Immovable blocks are placed everywhere except at $(-8,0), (0,0), (-8,8), (0,8), (-8,16), (0,16), (-8,24), (0,24)$. The raised platform is absent in the region $[-4, 12] \times [12, 20]$. A movable block is placed at $(8,8)$. The agent is initialized at position $(0,0,4.5)$. At each episode, the target position is fixed to $(g_x, g_y, g_z) = (0, 27, 4.5)$. Therefore, to cross the chasm, the agent must first push the movable block into the chasm and walk on top of it before navigating to the target.

At evaluation time, we evaluate the agent on its ability to reach $(0, 27, 4.5)$. We define a "success" as being within an L2 distance of 5 from the target on the ultimate step of the episode.

C Implementation Details

C.1 Network Structure

We use the same basic network structure as proposed by the TD3 algorithm [12], with the only difference being that we use layers of size $(300, 300)$ rather than $(400, 300)$.

The output of the lower-level actor network (activated by $\tanh$) is scaled to the range of the low-level actions, which is ± 30.

The output of the higher-level actor network is scaled to an approximated range of high-level actions: ± 10 for the desired relative x, y; ± 0.5 for the desired relative z; ± 1 for the desired relative torso orientations; and the remaining limb angle ranges are available from the ant.xml file.

C.2 Training Parameters

- Discount $\gamma = 0.99$ for both controllers.
- Adam optimizer; actor learning rate 0.0001; critic learning rate 0.001.
- Soft update targets $\tau = 0.005$ for both controllers.
- Replay buffer of size 200,000 for both controllers.
- Lower-level train step and target update performed every 1 environment step.
- Higher-level train step and target update performed every 10 environment steps.
- No gradient clipping.
- Reward scaling of 1.0 for lower-level; 0.1 for higher-level.
- Lower-level exploration is Gaussian noise with $\sigma = 1.0$.
- Higher-level exploration is Gaussian noise with $\sigma = 1.0$.

14

Ant Gather

| HIRO | VIME | SNN4HRL |
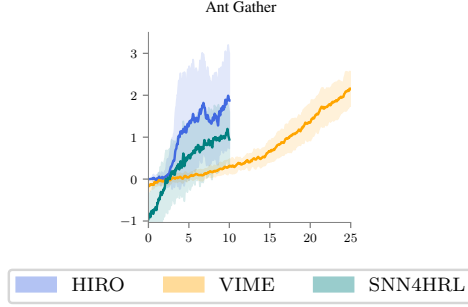
Figure 5: Performance of HIRO compared to VIME and SNN4HRL, averaged over 10 trials with x-axis in millions of experience samples. After a hyper-parameter search on the baselines, we were only able to get competitive performance with HIRO from VIME on Ant Gather, with a significantly higher amount of experience. On the other tasks, we were unable to achieve good baseline performance, even with more experience. The SNN4HRL curve does not include 25M transitions used in pre-training.

C.3 Off-Policy Correction

Given a high-level experience transition $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$, we select 10 candidate $\tilde{g}_t$ to maximize the log-probability of the lower-level actions. One is taken to be the original g_t; another to be $s_{t+c} - s_t$; and the remaining eight are sampled randomly from a Gaussian centered at $s_{t+c} - s_t$ with standard deviation $0.5 \times \frac{1}{2}$[high-level action range] (and subsequently clipped to lie within the high-level action range).

C.4 Evaluation

Learned hierarchical policies are evaluated every 50,000 training steps by averaging performance over 50 random episodes.

D Benchmark Details

D.1 FuN

FuN [48] primarily proposes four components: (1) transition policy gradient, (2) directional cosine similarity rewards, (3) goals specified with respect to a learned representation, and (4) dilated RNN. Since our tasks are low-dimensional and fully observed, we do not include design choice (4). For each of (1), (2), and (3), we apply an equivalent modification of our HRL method and evaluate its performance on the same tasks. For representation learning, we augment our method with a two-hidden-layer feed-forward neural network for embedding the observations before passing them to the lower and higher-level policies. The higher-level policy specifies high-level actions and rewards low-level behavior with respect to this representation. For the transition policy gradient, we modify our off-policy correction to instead replace a goal g_t with a goal $\tilde{g}_t$ sampled from a Gaussian centered at $s_{t+c} - s_t$, with standard deviation set to $0.1 \times \frac{1}{2}$[high-level action range]. This is analogous to FuN's transition policy gradient, which trains the higher-level policy under the assumption that its state transitions are distributed symmetrically around its proposed goals. For directional rewards, we replace our relative position parameterized reward function r with a cosine similarity reward function equivalent to that used in FuN.

D.2 SNN4HRL

SNN4HRL [10] trains the hierarchical policy stage-wise. It first uses a proxy reward with mutual information (MI) bonus to learn a mixture of low-level skills encouraging diverse movements, and then trains a high-level policy that controls switching among these skills to optimize for the task reward. We imported our additional environments to the official open-source codes with minimal modifications and followed the similar benchmark setup as in [10]. All policies are trained with

15
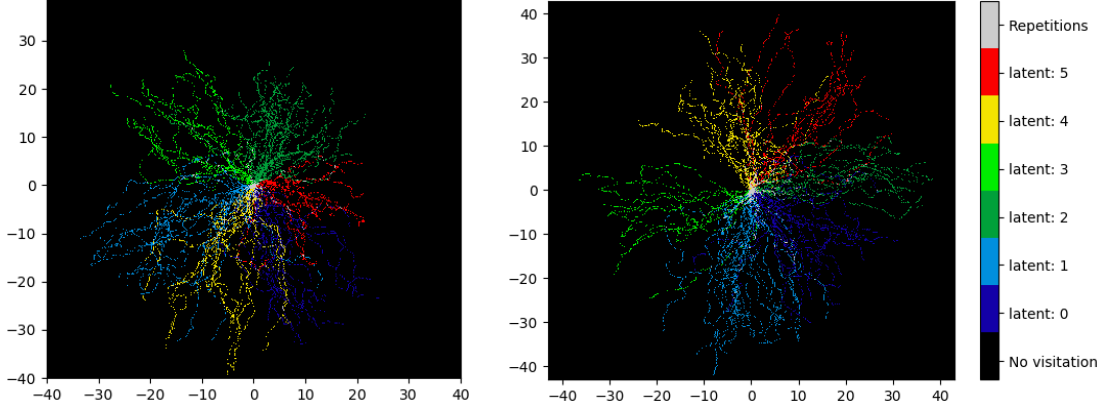
Figure 6: Visitation plots for 2 random seeds for the low-level SNN policy in the SNN4HRL benchmark. All 6 policies diversify in different exploration directions.

TRPO [39] with step size 0.01 and discount 0.99. All neural networks (the SNN, the Latent Regressor Network and the Manager Network) have 2 layers of 32 hidden units as done in [10][3]. We report the final results averaged over 10 random seeds, where we ran the low-level training using 5 seeds, and ran the high-level training using 2 seeds per trained low-level policy.

For the skill training, the mesh density used to grid the (x, y) space and give the MI bonus is 10 divisions/unit, and the mutual information bonus coefficient α_H is set to 10. Bilinear integration is used for the SNN. The number of skills trained is 6. The batch size and the maximum path length for the pre-train task are 50,000 and 500 respectively. To guarantee learning effective skills, we trained the policy with 500 updates, or 25M transitions. The visualization of learned skill primitives is shown in Figure 6, where the learned 6 sub-policies successfully cover different exploration directions.

For the high-level policy learning[4], we fixed the low-level policies as done in SNN4HRL and trained a policy that outputs a categorical action every 10 time steps that choose which skill to execute. For fair comparisons, we experimented with both sparse and dense rewards for the maze environments, and searched over batch sizes for $(1e4, 5e4, 5e5)$ transitions. We observed that the dense rewards did not help for SNN4HRL significantly, since the policy often quickly converge to local optimum. We found the batch size of $1e4$ is too noisy, and the batch size of $5e5$ is unnecessarily sample intensive, so the high-level policy is trained using batch size of $5e4$, the default value in their paper, for 300 updates, or 15M transitions. The combined training sample size of 40M is generously more than 10M used for our methods; however, our method still outperforms these SNN4HRL results substantially.

D.3 Variational Information Maximizing Exploration

Variational Information Maximizing Exploration (VIME) [21], while not a HRL algorithm, exhibits good performance on prior benchmark maze and gather tasks, and is also used as a strong baseline in SNN4HRL [10]. We ran the algorithm using the default settings in the official open-source implementation. Batch size of 50,000 is used. We report the average performance across 5 seeds after running the algorithm for 300 updates, or 15M transitions. Only the Gather task required more samples to converge to the final performance, and required 25M+ transitions to reach the same performance as what our method reached in a few million transitions.

---

[3]While the policy network sizes are significantly smaller than those used for our method, we observed no significant improvements with larger network sizes and this observation conforms with prior results that on-policy policy gradient methods can perform well on MuJoCo benchmark tasks with very small networks [9, 37].

[4]In both SNN4HRL [10] and VIME [21], primarily the results are reported and compared on SwimmerMaze and SwimmerGather, and therefore the experimental results are different.

16

D.4 Option-Critic Architecture

We also experimented with continuous-action variants of the option-critic architecture [2]. The option-policy $\pi_{\omega,\theta}(a|s)$ for option ω is parameterized as a Gaussian, whose mean is output from a neural network taking in s and ω, and variance is chosen to be global and diagonal. We first tested naively extending the official open-source implementation for continuous action, and then tried modifying the learning procedure such that the critic learns the state-option-action value function $Q_U(s,\omega,a)$ instead of the state-option value function $Q_\Omega(s,\omega)$ in the original implementation. This creates slight changes for the value and policy training objectives, while the loss for termination policy $\beta_{\omega,\nu}(s)$ is basically kept the same. Concretely, for the first variant, we trained $Q_\Omega(s,\omega)$ and the option-policy $\pi_{\omega,\theta}(a|s)$ with the following gradients,

$$g_\Omega = \mathbb{E}_{s_t,\omega_t,s_{t+1}\sim\beta}\left[\frac{\partial}{\partial\Omega}\left(Q_\Omega(s_t,\omega_t) - y_t\right)^2\right] \tag{21}$$

$$g_\theta = \mathbb{E}_{s_t,\omega_t,a_t,s_{t+1}\sim\pi}\left[(y_t - b_t)\nabla_\theta\log\pi_{\omega_t,\theta}(a_t|s_t)\right] \tag{22}$$

$$y_t = r_{t+1} + \gamma\left((1 - \beta_{\omega_t,\nu}(s_{t+1}))Q'(s_{t+1},\omega_t) + \beta_{\omega_t,\nu}(s_{t+1})\max_\omega Q'(s_{t+1},\omega)\right) \tag{23}$$

where Q' represents the target network, and β and π represent using off-policy and on-policy transition samples respectively. For simplicity of explanation, we assumed that the reward only depends on states, but similar arguments can be made for the general case. There are two pragmatic problems for this objective. First, the policy gradient, which relies on a score function estimate, could be high variance especially with respect to a continuous policy $\pi_{\omega,\theta}$. We experimented with several choices of baselines b_t, including $Q_\Omega(s_t,\omega_t)$ and $Q; (s_t,\omega_t)$. The second problem is that the off-policy learning for $Q_\Omega(s_t,\omega_t)$ does not use the action a_t taken and only relies on ω_t. This effectively creates the same non-stationarity problem with respect to the high-level policy as our method, since it ignores that for the same ω_t and s_t, the next state s_{t+1} can be different due to changing $\pi_{\omega,\theta}$. To counter both problems, we also explored another variant of the option-critic implementation at the expense of potentially more computation and network parameters, which conforms more closely with the policy gradient theorems in the original paper. Specifically, we trained $Q_U(s,\omega,a)$ and the option-policy $\pi_{\omega,\theta}(a|s)$ with the following gradients,

$$g_U = \mathbb{E}_{s_t,\omega_t,a_t,s_{t+1}\sim\beta}\left[\frac{\partial}{\partial U}\left(Q_U(s_t,\omega_t,a_t) - y_t\right)^2\right] \tag{24}$$

$$g_\theta = \mathbb{E}_{s_t,\omega_t\sim\pi}\left[\nabla_\theta\mathbb{E}_{a\sim\pi_{\omega_t,\theta}(a|s_t)}\left[Q_U(s_t,\omega_t,a)\right]\right] \tag{25}$$

$$y_t = r_{t+1} + \gamma\left((1 - \beta_{\omega_t,\nu}(s_{t+1}))Q'(s_{t+1},\omega_t) + \beta_{\omega_t,\nu}(s_{t+1})\max_\omega Q'(s_{t+1},\omega)\right) \tag{26}$$

$$Q'(s,\omega) = \mathbb{E}_{a\sim\pi_{\omega,\theta}(a|s)}\left[Q'(s,\omega,a)\right]. \tag{27}$$

In this implementation, we observe that the off-policy learning for $Q_U(s,\omega,a)$ can effectively utilize both ω_t and a_t, removing the non-stationarity problem, and the policy gradient can be estimated with lower variance using reparametrization trick [22] through the critic directly. Furthermore, since the policy gradient no longer requires next state estimate, off-policy state samples may also be used along with enumeration over all ω,

$$g_\theta = \mathbb{E}_{s_t\sim\beta}\left[\sum_\omega\nabla_\theta\mathbb{E}_{a\sim\pi_{\omega,\theta}(a|s_t)}\left[Q_U(s_t,\omega,a)\right]\right]. \tag{28}$$

Making similar approximations for the termination policy, this enables a fully off-policy actor-critic algorithm like DDPG [27] for the option-critic architecture.

While we tried these modifications, we could not make the option-critic implementation work reasonably on our domains. The main difficulty is likely because the low-level option-policies are learned using only the external task reward, a limitation in a direct end-to-end hierarchical policy structure. While in our experiments we could not show substantial successes, the algorithm may work better with more sophisticated modifications to the policy evaluation or policy improvement routines based on recent advances [30, 49, 14, 16, 12], and we leave further comparisons for future work.

17