HIERARCHICAL REINFORCEMENT LEARNING BY DISCOVERING INTRINSIC OPTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a hierarchical reinforcement learning method, HIDIO, that can learn task-agnostic options in a self-supervised manner while jointly learning to utilize them to solve sparse-reward tasks. Unlike current hierarchical RL approaches that tend to formulate a goal-reaching framework or pre-define ad hoc lower-level policies, HIDIO encourages lower-level option learning that is independent of the task at hand, requiring no assumptions or knowledge about the task structure. These options are learned through an intrinsic entropy minimization objective conditioned on the option trajectories. The learned options are diverse, shortsighted, and non task-driven. In experiments on robotic manipulation and navigation tasks with sparse rewards, HIDIO achieves higher success rates with greater sample efficiency than regular RL baselines and two state-of-the-art hierarchical RL methods.

1 INTRODUCTION

Imagine a wheeled robot learning to kick a soccer ball into a goal with sparse reward supervision. In order to succeed, it must discover how to first navigate in its environment, then touch the ball, and finally kick it into the goal, only receiving a positive reward at the end for completing the task. This is a naturally difficult problem for traditional reinforcement learning (RL) to solve, unless the task has been decomposed into temporally extended stages where each stage constitutes a much easier subtask. In this paper we ask, how do we learn such decompositions automatically and utilize them to solve sparse reward problems?

Deep RL has made great strides solving a variety of tasks recently, with hierarchical RL (hRL) demonstrating promise in solving such sparse reward tasks (Sharma et al., 2019b; Le et al., 2018; Merel et al., 2019; Ranchod et al., 2015). In hRL, the task is decomposed into a hierarchy of subtasks, where policies at the top of the hierarchy call upon policies below to perform actions to solve their respective subtasks. This abstracts away actions for the policies at the top levels of the hierarchy. hRL makes exploration easier by potentially reducing the number of steps the agent needs to take to explore its state space. Moreover, at higher levels of the hierarchy, temporal abstraction results in more aggressive, multi-step value bootstrapping when temporal-difference (TD) learning is employed. These benefits are critical in sparse reward tasks as they allow an agent to more easily discover reward signals and assign credit. However, many existing methods make assumptions about the task structure (*e.g.*, two stages: first moving towards the object, and then picking it up), and/or the skills needed to solve the task (*e.g.*, pre-programmed motor skills) (Florensa et al., 2016; Riedmiller et al., 2018; Lee et al., 2019; Hausman et al., 2018; Lee et al., 2020; Sohn et al., 2018; Ghavamzadeh & Mahadevan, 2003; Nachum et al., 2018). Instead, we seek to formulate a general method that can learn these abstractions, for any task, with little manual design.

The main contribution of this paper is HIDIO (**HI**erarchical RL by **D**iscovering **I**ntrinsic **O**ptions), a hierarchical method that discovers task-agnostic intrinsic options to be executed in the environment, in a self-supervised manner. These options are simultaneously learned to be applied to the environment to maximize reward. We propose and empirically evaluate various instantiations of the option discovery process, comparing the resulting options with respect to their final task performance. We demonstrate that HIDIO is able to efficiently learn and discover diverse options to be utilized for higher task reward with superior sample efficiency compared to other hierarchical methods.

2 PRELIMINARIES

We consider the reinforcement learning (RL) problem in a Markov Decision Process (MDP). Let $\mathbf{s} \in \mathbb{R}^S$ be the agent state. We use the terms "state" and "observation" interchangeably to denote the environment input to the agent. A state can be fully or partially observed. Without loss of generality, we assume a continuous action space $\mathbf{a} \in \mathbb{R}^A$ for the agent. Let $\pi_\theta(\mathbf{a}|\mathbf{s})$ be the policy distribution with learnable parameters θ, and $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ the transition probability that measures how likely the environment transitions to $\mathbf{s}_{t+1}$ given that the agent samples an action by $\mathbf{a}_t \sim \pi_\theta(\cdot|\mathbf{s}_t)$. After the transition to $\mathbf{s}_{t+1}$, the agent receives a deterministic scalar reward $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.

The objective of RL is to maximize the sum of discounted rewards with respect to θ:

$$\mathop{\mathbb{E}}_{\pi_\theta, \mathcal{P}} \left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \right] \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor. We will omit $\mathcal{P}$ in the expectation for notational simplicity.

In the options framework (Sutton et al., 1999), the agent can switch between different options during an episode, where an option is translated to a sequence of actions by an option-conditioned policy with a termination condition. For a two-level hierarchy, the expectation in Eq. 1 is taken over policies at both levels. In the following, we will use the terms "goal", "option", and "skill" interchangeably.

3 HIERARCHICAL RL BY DISCOVERING INTRINSIC OPTIONS

We now introduce our hierarchical method for solving sparse reward tasks in general settings. We allow the reward function to be defined in any form, not necessarily defined by an "ϵ-region" in the state space as assumed in some sparse reward settings (Andrychowicz et al., 2017; Riedmiller et al., 2018). For the purpose of this paper, we consider a hierarchy of two levels. The higher-level policy (the *scheduler* π_θ), is trained to maximize environment reward, while the lower-level policy (the *worker* π_ϕ) is trained in a self-supervised manner to efficiently discover options that are utilized by π_θ to accomplish tasks. Importantly, the worker gets access to dense intrinsic rewards regardless of the sparsity of the extrinsic rewards.

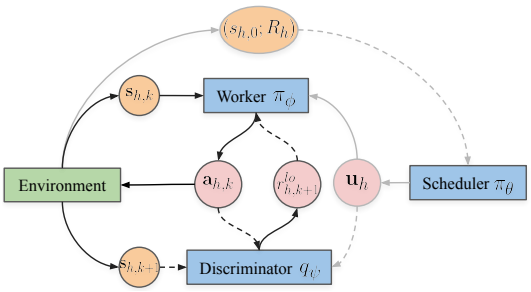

Figure 1: The overall framework of HIDIO. Solid lines denote rollout sampling while dashed lines denote training. Light-color lines denote a frequency of $\frac{1}{K}$, where K is the interval between two options. Refer to Eq. 2 for sampling and Eqs. 3 and 5 for training.

Without loss of generality, we assume that each episode has a length of T and the scheduler outputs an option every K steps. Given a scheduled option $\mathbf{u} \in [-1, 1]^D$ (*e.g.*, `tanh` activated), the worker executes K steps before the scheduler outputs the next option. Let the time horizon of the scheduler be $H = \lceil \frac{T}{K} \rceil$, and denote $\mathbf{s}_{h,k} \equiv \mathbf{s}_{hK+k}$ and $\mathbf{a}_{h,k} \equiv \mathbf{a}_{hK+k}$. Formally, we define

$$\begin{array}{lll} \text{Scheduler policy:} & \mathbf{u}_h \sim \pi_\theta(\cdot|\mathbf{s}_{h,0}), & 0 \le h < H \\ \text{Worker policy:} & \mathbf{a}_{h,k} \sim \pi_\phi(\cdot|\mathbf{s}_{h,k}, \mathbf{u}_h), & 0 \le k < K \\ \text{Environment dynamics:} & \mathbf{s}_{t+1} \sim \mathcal{P}(\cdot|\mathbf{s}_t, \mathbf{a}_t), & 0 \le t < T \end{array} \tag{2}$$

Note that given the notation and sampling process, we have $\mathbf{s}_{h,K} \equiv \mathbf{s}_{h+1,0}$, namely, the last state of the current option $\mathbf{u}_h$ is the initial state of the next option $\mathbf{u}_{h+1}$. The overall framework of our method is illustrated in Figure 1. The pseudo code of HIDIO is in Appendix A.

3.1 LEARNING THE SCHEDULER

Every time the scheduler issues an option $\mathbf{u}_h$, it receives an reward R_h computed by accumulating environment rewards over the next K steps. Its objective is:

$$\max_{\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{h=0}^{H-1} \beta^h R_h\right], \quad \text{where} \quad \beta = \gamma^K \text{ and } R_h = \mathbb{E}_{\pi_\phi}\left[\sum_{k=0}^{K-1} \gamma^k r(\mathbf{s}_{h,k}, \mathbf{a}_{h,k}, \mathbf{s}_{h,k+1})\right] \tag{3}$$

This scheduler objective itself is not a new concept, as similar ones have been adopted by other hRL methods (Vezhnevets et al., 2017; Nachum et al., 2018; Riedmiller et al., 2018). One significant difference between our option with that of prior work is that there is no constraint on what $\mathbf{u}$ could represent. In contrast, existing methods usually require their options to reside in a subspace of the state space, to be grounded to the environment, or to have known structures, so that the scheduler can compute rewards and termination conditions for the worker. As a result, their options are task-dependent and might have difficulty adapting to new scenarios without being redesigned.

3.2 Learning the Worker

The main focus of this paper is to investigate how to effectively learn the worker policy in a self-supervised manner. Many existing hRL works either pre-define or pre-train ad hoc worker policies (Riedmiller et al., 2018) to be composed by the scheduler, or jointly train the worker to reach goals that reside in a subspace of the (latent) state space (Vezhnevets et al., 2017; Nachum et al., 2018). Our motivation is that it might be unnecessary to explicitly ask the worker to reach some "ϵ-space" of goals. As long as a goal can be translated to a short sequence of primitive actions, it does not need to be grounded with concrete meanings. Our worker learning is inspired by similar ideas explored by DIAYN (Eysenbach et al., 2019) and DADS (Sharma et al., 2019a) for unsupervised skill discovery, but their goals are fixed over entire episodes. Also different from them, our worker goes beyond using individual state features alone for identifying options.

Worker Objective We first define a new MDP so that for any h, k, and t:

1) $\bar{\mathbf{s}}_{h,k} := (\mathbf{s}_{h,0}, \ldots, \mathbf{s}_{h,k})$,
2) $\bar{\mathbf{a}}_{h,k} := (\mathbf{a}_{h,0}, \ldots, \mathbf{a}_{h,k})$,
3) $r(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t, \bar{\mathbf{s}}_{t+1}) := r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, and
4) $\mathcal{P}(\bar{\mathbf{s}}_{t+1}|\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t) := \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$.

This new MDP equips the worker with historical state and action information since an option was scheduled. In the following, we will call pairs $\{\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}\}$ option *sub-trajectories*. The worker policy now takes option sub-trajectories as inputs: $\mathbf{a}_{h,k} \sim \pi_\phi(\cdot|\bar{\mathbf{s}}_{h,k}, \bar{\mathbf{a}}_{h,k-1}, \mathbf{u}_h), 0 \leq k < K$, whereas the scheduler policy still operates in the original MDP.

Denote $\sum_{h,k} \equiv \sum_{h=0}^{H-1}\sum_{k=0}^{K-1}$ for simplicity. The worker objective is defined on this new MDP, to minimize the entropy of the option $\mathbf{u}_h$ conditioned on the option sub-trajectory $\{\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}\}$:

$$\max_{\phi} \mathop{\mathbb{E}}_{\pi_\theta, \pi_\phi} \sum_{h,k} \underbrace{\log p(\mathbf{u}_h|\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})}_{\text{negative conditional option entropy}} \underbrace{-\beta \log \pi_\phi(\mathbf{a}_{h,k}|\bar{\mathbf{s}}_{h,k}, \bar{\mathbf{a}}_{h,k-1}, \mathbf{u}_h)}_{\text{worker policy entropy}} \tag{4}$$

where the expectation is over the current π_θ and π_ϕ but the maximization is only with respect to ϕ. Intuitively, the first term suggests that the worker is optimized to confidently identify an option given a sub-trajectory. However, it alone will not guarantee the diversity of options because potentially even very similar sub-trajectories can be mapped to different options if the resulting sub-trajectory space has a very high "resolution". As a result, the conditional entropy might not be able to generate useful skills to be exploited by the scheduler for task solving. To combat this degenerate solution, we add a second term which computes the entropy of the worker policy. Intuitively, while the worker generates identifiable sub-trajectories corresponding to a given option, it should act as randomly as possible to separate sub-trajectories of different options, lowering the "resolution" of the sub-trajectory space to encourage diversity. Because directly estimating the posterior $p(\mathbf{u}_h|\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})$ is intractable, we approximate it with a parameterized posterior $\log q_\psi(\mathbf{u}_h|\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})$ to obtain a lower bound (Barber & Agakov, 2003), where q_ψ is a *discriminator* to be learned. Then we can maximize this lower bound instead:

$$\max_{\phi,\psi} \mathbb{E}_{\pi_\theta, \pi_\phi} \sum_{h,k} \log q_\psi(\mathbf{u}_h|\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}) - \beta \log \pi_\phi(\mathbf{a}_{h,k}|\bar{\mathbf{s}}_{h,k}, \bar{\mathbf{a}}_{h,k-1}, \mathbf{u}_h). \tag{5}$$

Similar to VIC (Gregor et al., 2016) and DIAYN (Eysenbach et al., 2019), the discriminator is trained by maximizing likelihoods of options given sampled sub-trajectories. The worker is trained via max-entropy RL (Soft Actor-Critic (SAC) (Haarnoja et al., 2018)) with the intrinsic reward $r^{lo}_{h,k+1} := \log q_\psi(\cdot) - \beta \log \pi_\phi(\cdot)$. β is a constant weight which is fixed to 0.01 in all our experiments.

Note that there are at least two differences between Eq. 5 and the skill discovery objectives in VIC (Gregor et al., 2016) and DIAYN (Eysenbach et al., 2019). First, both VIC and DIAYN assume that a sampled skill will last through an entire episode, and the skill is always sampled at the beginning of an episode. Thus their skill trajectories "radiate" from the initial state set. In contrast, our worker policy learns skills that are reset every K steps within an episode, and they can have more diverse semantics depending on the various states visited by the agent. This is especially helpful for some tasks where new skills need to be discovered after the agent reaches unseen areas in later stages of training. Second, we extend the discriminator to predict based on option sub-trajectories $\{\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}\}$ instead of solely on individual states $\mathbf{s}_{h,k}$, and we investigate six possible instantiations of the discriminator. As observed in the experiments, individual states are actually not the optimal features for identifying options.

Shortsighted Worker It's challenging for the worker to accurately predict values over a long horizon, since its rewards are densely computed by a complex nonlinear function q_ψ. Also each option only lasts at most K steps. Thus we set the discount η for the worker in two shortsighted ways:

1. `Hard`: $\eta = 0$ if $t = hK$ and $\eta = 1$ otherwise. Basically this version truncates the temporal correlation (gradients) between adjacent options. The benefit of this might be faster and easier value learning because the value is bootstrapped over at most K steps ($K \ll T$).
2. `Soft`: $\eta = 1 - \frac{1}{K}$, which considers rewards of roughly K steps ahead. The worker policy still needs to take into account the identification of future option sub-trajectories, but their importance quickly decays.

We will evaluate both versions and compare their performance in Section 4.1.

3.3 Instantiating the Discriminator

We conduct a comprehensive study of various ways of instantiating the discriminator q_ψ in order to compute useful intrinsic rewards for the worker. Previous work has utilized individual states (Eysenbach et al., 2019; Jabri et al., 2019) or full observation trajectories (Warde-Farley et al., 2019; Sharma et al., 2019a; Achiam et al., 2018) for skill discrimination. Because our discriminator takes option sub-trajectories for prediction, in principle we can instantiate our discriminator on any function of an option sub-trajectory. In this paper, we investigate six sub-trajectory feature extractors f_ψ:

Feature extractor	Name	Formulation	Explanation
$f_\psi(\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}) =$	`State`	$\textsc{Mlp}(\mathbf{s}_{t,k+1})$	Next state alone
	`Action`	$\textsc{Mlp}([\mathbf{s}_{h,0}, \mathbf{a}_{h,k}])$	Action in context
	`StateDiff`	$\textsc{Mlp}(\mathbf{s}_{h,k+1} - \mathbf{s}_{h,k})$	Difference between state pairs
	`StateAction`	$\textsc{Mlp}([\mathbf{a}_{h,k}, \mathbf{s}_{h,k+1}])$	Action and next state
	`StateConcat`	$\textsc{Mlp}([\bar{\mathbf{s}}_{h,k+1}])$	Concatenation of states
	`ActionConcat`	$\textsc{Mlp}([\mathbf{s}_{h,0}, \bar{\mathbf{a}}_{h,k}])$	Concatenation of actions

where the operator $[\cdot]$ denotes concatenation and $\textsc{Mlp}$ denotes a multilayer perceptron[1]. Our `State` feature extractor is most similar to DIAYN (Eysenbach et al., 2019), and `StateConcat` is similar to (Warde-Farley et al., 2019; Sharma et al., 2019a; Achiam et al., 2018). However we note that unlike these works, the reachability of our option sub-trajectories is determined by the scheduler in the context of hRL. The other four feature extractors have not been evaluated before. With the extracted feature, the log-probability of predicting an option is simply computed as the negative squared L2 norm: $\log q_\psi(\mathbf{u}_h | \bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}) = -\| f_\psi(\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}) - \mathbf{u}_h \|_2^2$, through which we implicitly assume the discriminator's output distribution to be a multi-variate Gaussian with zero means and unit variances.

[1] In this paper we focus on non-image observations that can be processed with MLPs, although our method doesn't have any assumption about the observation space.

Figure 2: The four tasks we evaluate on. From left to right: 7-DOF PUSHER, 7-DOF REACHER, GOALTASK, and KICKBALL. The first two tasks simulate a one-armed PR2 robot environment while the last two are in the SOCIALROBOT environment. The final picture shows a closeup of the PIONEER2DX robot used in SOCIALROBOT.

3.4 OFF-POLICY TRAINING

The scheduler and worker objectives (Eq. 3 and Eq. 5) are trained jointly. In principle, on-policy training such as A2C (Clemente et al., 2017) is needed due to the interplay between the scheduler and worker. However, to reuse training data and improve sample efficiency, we employ off-policy training (SAC (Haarnoja et al., 2018)) for both objectives with some modifications.

Modified worker objective In practice, the expectation over the scheduler π_θ in Eq. 5 is replaced with the expectation over its historical versions. Specifically, we sample options $\mathbf{u}_h$ from a replay buffer, together with sub-trajectories $\{\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1}\}$. This type of data distribution modification is conventional in off-policy training (Lillicrap et al., 2016).

Intrinsic reward relabeling We always recompute the rewards in Eq. 5 using the up-to-date discriminator for every update of ϕ, which can be trivially done without any additional interaction with the environment.

Importance correction The data in the replay buffer was generated by historical worker policies. Thus a sampled option sub-trajectory will be outdated under the same option, causing confusion to the scheduler policy. To resolve this issue, when minimizing the temporal-difference (TD) error between the values of $\mathbf{s}_{h,0}$ and $\mathbf{s}_{h+1,0}$ for the scheduler, an importance ratio can be multiplied: $\prod_{k=0}^{K-1} \frac{\pi_\phi(\mathbf{a}_{h,k}|\mathbf{s}_{h,k},\mathbf{u}_h)}{\pi_\phi^{old}(\mathbf{a}_{h,k}|\mathbf{s}_{h,k},\mathbf{u}_h)}$. A similar correction can also be applied to the discriminator loss. However, in practice we find that this ratio has a very high variance and does not help. Empirically, even without correction our method is able to perform well[2]; similar observations were made in Nachum et al. (2018); Fedus et al. (2020). We leave further reduction of off-policy biases to future work.

4 EXPERIMENTS

Environments We evaluate success rate and sample efficiency across two environment suites, as shown in Figure 2. Important details are presented here with more information in appendix Section B. The first suite consists of two 7-DOF reaching and pushing environments evaluated in Chua et al. (2018). They both emulate a one-armed PR2 robot. The tasks have sparse rewards: the agent gets a reward of 0 at every timestep where the goal is not achieved, and 1 upon achieved. There is also a small L_2 action penalty applied. In 7-DOF REACHER, the goal is achieved when the gripper reaches a 3D goal position. In 7-DOF PUSHER, the goal is to push an object to a 3D goal position. A success of an episode is defined to be if the goal is achieved at the *final* step of the episode.

We also propose another suite of environments called SOCIALROBOT. We construct two sparse reward robotic navigation and manipulation tasks, GOALTASK and KICKBALL. In GOALTASK, the agent gets a reward of 1 when it successfully navigates to a goal, -1 if the goal becomes too far, -0.5 every time it is too close to a distractor object, and 0 otherwise. In KICKBALL, the agent receives a reward of 1 for successfully pushing a ball into the goal, 0 otherwise, and has the same distractor object penalty. At the beginning of each episode, both the agent and the ball are spawned randomly. Both environments contain a small L_2 action penalty, and terminate an episode upon a success.

Comparison methods One baseline algorithm for comparison is standard SAC (Haarnoja et al., 2018), the building block of our hierarchical method. To verify if our worker policy can just be re-

[2]One possible reason is that the deep RL process is "highly non-stationary anyway, due to changing policies, state distributions and bootstrap targets" (Schaul et al., 2016).

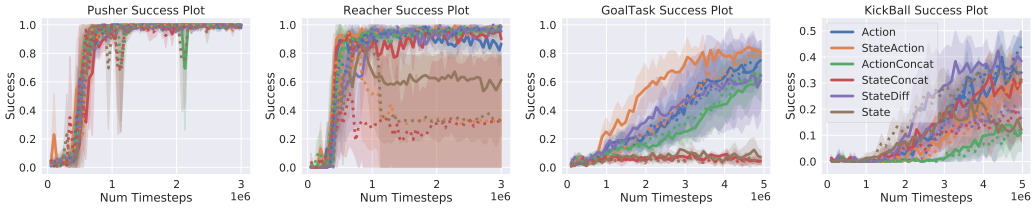

Figure 3: Comparison of all HIDIO discriminator features against each other across the four environments. Solid lines indicate hard short-sighted workers (`Hard`), dotted lines indicated soft short-sighted workers (`Soft`).

placed with a naïve action repetition strategy, we compare with SAC+ActRepeat with an action repetition for the same length K as our option interval. We also compare against HIRO (Nachum et al., 2018), a data efficient hierarchical method with importance-based option relabeling, and HiPPO (Li et al., 2020) which trains the lower level and higher level policies together with one unified PPO-based objective. Both are state-of-the-art hierarchical methods proposed to solve sparse reward tasks. Similar to our work, HiPPO makes no assumptions about options, however it utilizes a discrete option space and its options are trained with environment reward.

A comprehensive hyperparameter search is performed for every method, with a far greater search space over HiPPO and HIRO than our method HIDIO to ensure maximum fairness in comparison; details are presented in Appendix C.

Evaluation For every evaluation point during training, we evaluate the agent with current deterministic policies (by taking $\arg\max$ of action distributions) for a fixed number of episodes and compute the mean success rate. We plot the mean evaluation curve over 3 randomly seeded runs with standard deviations shown as the shaded area around the curve.

4.1 Worker Design Choices

We ask and answer questions about the design choices in HIDIO specific to the worker policy π_ϕ.

1. *What sub-trajectory feature results in good option discovery?* We evaluate all six features proposed in Section 3.3 in all four environments. These features are selected to evaluate how different types of subtrajectory information affect option discovery and final performance. They encompass varying types of both local and global subtrajectory information. We plot comparisons of sample efficiency and final performance in Figure 3 across all environments (solid lines), finding that `Action`, `StateAction`, and `StateDiff` are generally among the top performers. `StateAction` includes the current action and next state, encouraging π_ϕ to differentiate its options with different actions even at similar states. Similarly, `Action` includes the option initial state and current action, encouraging option diversity by differentiating between actions conditioned on initial states. Meanwhile `StateDiff` simply encodes the difference between the next and current state, encouraging π_ϕ to produce options with different state changes at each step.

2. *How do soft shortsighted workers (`Soft`) compare against hard shortsighted workers (`Hard`)?* In Figure 3, we plot all features with `Soft` in dotted lines. We can see that in general there is not much difference in performance between `Hard` and `Soft` except some extra instability of `Soft` in Reacher regarding the `StateConcat` and `State` features. One reason of this similar general performance could be that since our options are very short-term in `Hard`, the scheduler policy has the opportunity of switching to a good option before the current one leads to bad consequences. In a few cases, `Hard` seems better learned, perhaps due to an easier value bootstrapping for the worker.

4.2 Comparison Results

We compare our three best sub-trajectory features of `Hard`, in Section 4.1, against the SAC baselines and hierarchical RL methods across all four environments in Figure 4. Generally we see that HIDIO (solid lines) achieves greater final performance with superior sample efficiency than the compared methods. Both SAC and SAC+ActRepeat perform poorly across all environments, and all baseline methods perform significantly worse than HIDIO on Reacher, GoalTask, and KickBall.

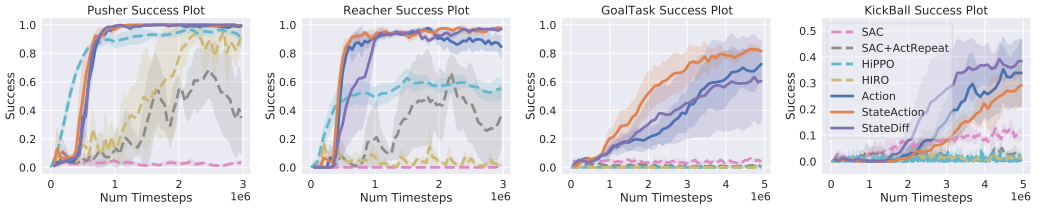

Figure 4: Comparisons of the mean success rates of three features of HIDIO (`Action`, `StateAction`, `StateDiff`; solid lines) against other methods (dashed lines).

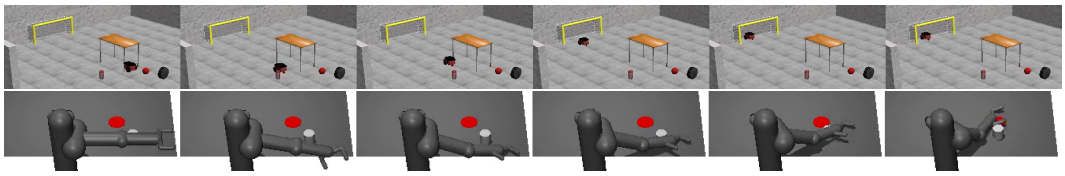

Figure 5: Two example options/skills from the `StateAction` HIDIO instantiation on KICKBALL (top) and PUSHER (bottom). The top skill navigates directly to the goal by bypassing obstructions along the way, and the bottom skill sweeps the puck towards one direction.

In PUSHER, HiPPO displays competitive performance, rapidly improving from the start. However, all three HIDIO instantiations achieve nearly 100% success rates while HiPPO is unable to do so. Furthermore, HIRO and SAC+ActRepeat take much longer to start performing well, but never achieve similar success rates as HIDIO. HIDIO is able to solve REACHER while HiPPO achieves only about a 60% success rate at best. Meanwhile, HIRO, SAC+ActRepeat, and SAC are unstable or non-competitive. REACHER is a difficult exploration problem as the arm starts far from the goal position, and we see that HIDIO's automatically discovered options ease exploration for the higher level policy to consistently reach the goal. HIDIO performs well on GOALTASK, achieving 60-80% success rates, while the task is too challenging for every other method. In KICKBALL, the most challenging task, HIDIO achieves 30-40% success rates while every other method fails to learn again, highlighting the need for the intrinsic option discovery of HIDIO in these environments.

In summary, HIDIO demonstrates greater sample efficiency and final reward gains over all other baseline methods. Regular RL (SAC) fails on all four environments, and while HiPPO is a strong baseline on PUSHER and REACHER, it is still outperformed in both by HIDIO. All other methods fail on GOALTASK and KICKBALL, while HIDIO is able to learn and perform better in both. This demonstrates the importance of the intrinsic, short-term option discovery employed by HIDIO, where the options are diverse enough to be useful for both exploration and task completion.

4.3 JOINT π_ϕ AND π_θ TRAINING

We ask the next question: *is jointly training π_θ and π_ϕ necessary?* To answer this, we compare HIDIO against a pre-training baseline where we first pre-train π_ϕ, with uniformly sampled options $\mathbf{u}$ for a portion ρ of total numbers of time steps, and then fix π_ϕ while training π_θ for the remaining $(1-\rho)$ time steps. This is essentially using pre-trained skills for downstream higher-level tasks as demonstrated in DIAYN (Eysenbach et al., 2019). We conduct this experiment with the `StateAction` feature extractor on both KICKBALL and PUSHER, with $\rho = \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}\}$. The results are shown in Figure 6. We can see that in PUSHER, fewer pretraining timesteps are more sample efficient, as the environment is simple and options can be learned from a small amount of samples. The nature of PUSHER also only requires stationary options that can be learned independent of the scheduler policy evolution. Nevertheless, the pretraining baselines seem less stable. In KICKBALL, the optimal pretraining baseline is on $\rho = \frac{1}{8}$ of the total timesteps. However without the joint training scheme of HIDIO, the learned

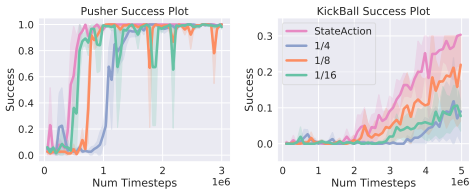

Figure 6: Pretraining baseline comparison at fractions $\{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}\}$ of the total number of timesteps.

skills are unable to be used as efficiently for the difficult obstacle avoidance, navigation, and ball manipulation subtasks required for performing well.

4.4 Option Behaviors

Finally, since options discovered by HIDIO in our sparse reward environments help it achieve superior performance, we ask, *what do useful options look like?* To answer this question, after training, we sample options from the scheduler π_θ to visualize their behaviors in different environments in Figure 5. For each sampled option $\mathbf{u}$, we fix it until the end of an episode and use the worker π_ϕ to output actions given $\mathbf{u}$. We can see that the options learned by HIDIO are low-level navigation and manipulation skills useful for the respective environments.

5 Related Work

Hierarchical RL Much of the previous work in hRL makes assumptions about the task structure and/or the skills needed to solve the task. While obtaining promising results under specific settings, they may have difficulties with different scenarios. For example, SAC-X (Riedmiller et al., 2018) requires manually designing auxiliary subtasks as skills to solve a given downstream task. SNN4HRL (Florensa et al., 2016) is geared towards tasks with pre-training and downstream components. Lee et al. (2019; 2020) learns to modulate or compose given primitive skills that are customized for their particular robotics tasks. Ghavamzadeh & Mahadevan (2003) and Sohn et al. (2018) operate under the assumption that tasks can be manually decomposed into subtasks.

The feudal reinforcement learning proposal (Dayan & Hinton, 1993) has inspired another line of works (Vezhnevets et al., 2017; Nachum et al., 2018; Levy et al., 2019) which make higher-level manager policies output goals for lower-level worker policies to achieve. Usually the goal space is a subspace of the state space or defined according to the task so that lower-level rewards are easy to compute. This requirement of manually "grounding" goals in the environment poses generalization challenges for tasks that cannot be decomposed into state or goal-reaching.

Our method HIDIO does not make any assumptions about the specific task at hand. It follows from the options framework (Sutton et al., 1999), which has recently been applied to continuous domains (Bacon et al., 2017), spawning a diverse set of recent hierarchical options methods (Bagaria & Konidaris, 2020; Klissarov et al., 2017; Riemer et al., 2018; Tiwari & Thomas, 2019; Jain et al., 2018). HIDIO automatically learns intrinsic options that avoids having explicit initiation or termination policies dependent on the task at hand. HiPPO (Li et al., 2020), like HIDIO, also makes no major assumptions about the task, leading to a more general hRL algorithm.

Self-supervised skill discovery There are also plenty of prior works which attempt to learn skills without task reward. DIAYN (Eysenbach et al., 2019) and VIC (Gregor et al., 2016) learn skills by maximizing the mutual information between trajectory states and their corresponding skills. VALOR (Achiam et al., 2018) learns options by maximizing the probability of options given their resulting observation trajectory. DADS (Sharma et al., 2019a) learns skills that are predictable by dynamics models. DISCERN (Warde-Farley et al., 2019) maximizes the mutual information between goal and option termination states to learn a goal-conditioned reward function. Eysenbach et al. (2019); Achiam et al. (2018); Sharma et al. (2019a) demonstrate these pre-trained skills to be useful for hRL. However, usually these methods pre-train skills in an initial stage that is separate from downstream task learning. Little work has been done to integrate these skill discovery processes into hRL.

6 Conclusion

Under the context of solving difficult sparse reward tasks, we propose a new hierarchical reinforcement learning method, HIDIO, which can learn task-agnostic options in a self-supervised manner and simultaneously learn to utilize them to solve tasks. We evaluate several different instantiations of the discriminator of HIDIO for providing intrinsic rewards for training the lower-level worker policy. We demonstrate the effectiveness of HIDIO compared against other reinforcement learning methods in achieving high rewards with better sample efficiency across a variety of robotic navigation and manipulation tasks.

REFERENCES

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv*, 2018. 4, 8

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NeurIPS*, 2017. 2

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017. 8

Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *ICLR*, 2020. 8

D. Barber and F. Agakov. The im algorithm: A variational approach to information maximization. In *NeurIPS*, 2003. 3

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018. 5

Alfredo V. Clemente, Humberto Nicolás Castejón Martínez, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *CoRR*, abs/1705.04862, 2017. 5

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *NeurIPS*, pp. 271–278, 1993. 8

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2019. 3, 4, 7, 8

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *ICML*, 2020. 5

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2016. 1, 8

Mohammad Ghavamzadeh and Sridhar Mahadevan. Hierarchical policy gradient algorithms. *ICML*, 2003. 1, 8

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv*, abs/1611.07507, 2016. 4, 8

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018. 4, 5

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *ICLR*, 2018. 1

Allan Jabri, Kyle Hsu, Ben Eysenbach, Abhishek Gupta, Sergey Levine, and Chelsea Finn. Unsupervised curricula for visual meta-reinforcement learning. In *NeurIPS*, 2019. 4

Arushi Jain, Khimya Khetarpal, and Doina Precup. Safe option-critic: Learning safety in the option-critic architecture. *arXiv*, 2018. 8

Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. *arXiv*, 2017. 8

Hoang M. Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. In *ICML*, 2018. 1

Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward Hu, and Joseph J. Lim. Composing complex skills by learning transition policies with proximity reward induction. In *ICLR*, 2019. 1, 8

Youngwoon Lee, Jingyun Yang, and Joseph J. Lim. Learning to coordinate manipulation skills via skill behavior diversification. In *ICLR*, 2020. 1, 8

Andrew Levy, Robert Platt Jr., and Kate Saenko. Learning multi-level hierarchies with hindsight. In *ICLR*, 2019. 8

Alexander C. Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. In *ICLR*, 2020. 6, 8

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. In *ICLR*, 2016. 5

Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. In *ICLR*, 2019. 1

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018. 1, 3, 5, 6, 8

Pravesh Ranchod, Benjamin Rosman, and George Konidaris. Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In *IROS*, 2015. 1

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom van de Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *ICML*, 2018. 1, 2, 3, 8

Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. In *NeurIPS*, 2018. 8

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR*, 2016. 5

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv*, abs/1907.01657, 2019a. 3, 4, 8

Arjun Sharma, Mohit Sharma, Nicholas Rhinehart, and Kris M. Kitani. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. In *ICLR*, 2019b. 1

Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In *NeurIPS*, 2018. 1, 8

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):181 – 211, 1999. 2, 8

Saket Tiwari and Philip S. Thomas. Natural option critic. In *AAAI*, 2019. 8

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017. 3, 8

David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised control through non-parametric discriminative rewards. In *ICLR*, 2019. 4, 8

A PSEUDO CODE FOR HIDIO

Algorithm 1: Hierarchical RL with Intrinsic Options Discovery

Input:

T	Episode length	M	Batches per iteration	$\pi_\phi(\mathbf{a}_{h,k}\|\mathbf{s}_{h,k}, \mathbf{u}_h)$	Worker
B	Batch size	α	Learning rate	$q_\psi(\mathbf{u}_h\|\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})$	Discriminator
K	Option interval	$\mathcal{P}(\mathbf{s}_{t+1}\|\mathbf{s}_t, \mathbf{a}_t)$	Environment dynamics	$\pi_\theta(\mathbf{u}_h\|\mathbf{s}_{h,0})$	Scheduler

Output: Learned parameters θ, ϕ, and ψ.
Initialize: Random model parameters θ, ϕ, and ψ; empty replay buffers $\mathcal{D}_{\text{scheduler}}$ and $\mathcal{D}_{\text{worker}}$.
while *termination not met* **do**
  `/* Data collection */`
  for *scheduler step* $h = 0..\frac{T}{K} - 1$ **do**
    Sample an option $\mathbf{u}_h \sim \pi_\theta(\cdot|\mathbf{s}_{h,0})$.
    for *worker step* $k = 0..K-1$ **do**
      Sample an action $\mathbf{a}_{h,k} \sim \pi_\phi(\cdot|\mathbf{s}_{h,k}, \mathbf{u}_h)$.
      Step through the environment $\mathbf{s}_{h,k+1} \sim \mathcal{P}(\cdot|\mathbf{s}_{h,k}, \mathbf{a}_{h,k})$.
      $\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1} \leftarrow [\bar{\mathbf{a}}_{h,k-1}, \mathbf{a}_{h,k}], [\bar{\mathbf{s}}_{h,k}, \mathbf{s}_{h,k+1}]$
      $\mathcal{D}_{\text{worker}} \leftarrow \mathcal{D}_{\text{worker}} \cup (\mathbf{u}_h, \bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})$
    end
    $R_h \leftarrow \sum_{k=0}^{K-1} r(\mathbf{s}_{h,k}, \mathbf{a}_{h,k}, \mathbf{s}_{h,k+1})$
    $\mathcal{D}_{\text{scheduler}} \leftarrow \mathcal{D}_{\text{scheduler}} \cup (\mathbf{s}_{h,0}, \mathbf{u}_h, \mathbf{s}_{h+1,0}, R_h)$
  end
  `/* Model training */`
  for *batch* $m = 0..M-1$ **do**
    `/* Scheduler training */`
    Uniformly sample transitions $\{(\mathbf{s}_t, \mathbf{u}_t, \mathbf{s}_{t+1})\}_{b=1}^{B} \sim \mathcal{D}_{\text{scheduler}}$.
    Compute gradient $\Delta\theta$ according to Eq. 3.
    Update models $\theta \leftarrow \theta + \alpha\Delta\theta$.
    `/* Worker training */`
    Uniformly sample transitions $\{(\mathbf{u}_h, \bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})\}_{b=1}^{B} \sim \mathcal{D}_{\text{worker}}$.
    Compute intrinsic rewards $r^{lo}_{h,k} \leftarrow q_\psi(\mathbf{u}_h|\bar{\mathbf{a}}_{h,k}, \bar{\mathbf{s}}_{h,k+1})$.
    Compute gradient $\Delta\psi$ and $\Delta\phi$ according to Eq. 5.
    Update models $\phi \leftarrow \phi + \alpha\Delta\phi$ and $\psi \leftarrow \psi + \alpha\Delta\psi$.
  end
end

B MORE ENVIRONMENT DETAILS

B.0.1 PUSHER AND REACHER

These environments both have a time horizon of 100 with no early termination: each episode always runs for 100 steps regardless of goal achievement. For both, a success is when the agent achieves the goal at the *final* step of an episode. In REACHER, observations are 17-dimensional, including the positions, angles, and velocities of the robot arm, and in PUSHER observations also include the 3D object position. Both include the goal position in the observation space. Actions are 7-dimensional vectors for joint velocity control. The action range is $[-20, 20]$ in REACHER and $[-2, 2]$ in PUSHER.

There is an action penalty in both environments: at every timestep the squared L_2 norm of the agent action is subtracted from the reward. In PUSHER, this penalty is multiplied by a coefficient of 0.001. In REACHER, it's multiplied by 0.0001.

B.0.2 GOALTASK AND KICKBALL

For both SOCIALROBOT environments, an episode terminates early when either a success is reached or the goal is out of range. For each episode, the positions of all objects (including the agent) are randomly picked. Observations are 18-dimensional, including ego-centric distances and directions from the agent to different objects. In KICKBALL, the agent receives a reward of 1 for successfully

pushing a ball into the goal (episode termination) and 0 otherwise. At the beginning of each episode, the ball is spawned randomly inside the neighborhood of the agent. Three distractor objects are included on the ground to increase task difficulty. In GOALTASK, the number of distractor objects increases to 5. Both environments contain a small L_2 action penalty: at every time step the squared L_2 norm of the agent action, multiplied by 0.01, is subtracted from the reward. GOALTASK has a time horizon of 100 steps, while KICKBALL's horizon is 200. Observations are 30-dimensional, including absolute poses and velocities of the goal, the ball, and the agent. Both GOALTASK and KICKBALL use the same navigation robot PIONEER2DX which has 2-dimensional actions that control the angular velocities (scaled to $[-1, 1]$) of the two wheels.

C HYPERPARAMETERS

To ensure a fair comparison across all methods, we perform a hyperparameter search over the following values for each algorithm and suite of environments.

C.1 PUSHER AND REACHER

Shared hyperparameters across all methods are listed below (where applicable, and except when overridden by hyperparameters listed for each individual method). For all methods, we take the hyperparameters that perform best across 3 random seeds in terms of the area under the evaluation success curve (AUC) in the PUSHER environment.

- Number of parallel actors/environments per rollout: 20
- Steps per episode: 100
- Batch size: 2048
- Learning rate: 10^{-4} for all network modules
- Policy/Q network hidden layers: (256, 256, 256) with ReLU non-linearities
- Polyak averaging coefficient for target Q: 0.999
- Target Q update interval (training iterations): 1
- Training batches per iteration: 100
- Episodes per evaluation: 50
- Initial environment steps for data collection before training: 10000

Rollouts and training iterations are performed alternatively, one after the other. The *rollout length* searched below refers to how many time steps in each environment are taken per rollout/training iteration, effectively controlling the ratio of gradient steps to environment steps. A smaller rollout length corresponds to a higher ratio. This ratio is also searched over for HIPPO and HIRO. Other hyperparameters searched separately for each algorithm are listed below, and selected ones are **bolded**.

C.1.1 SAC

- Target entropy min prob Δ[3]: {0.1, **0.2**, 0.3}
- Replay buffer length per parallel actor: {50000, **200000**}
- Rollout Length: {12, 25, **50**, 100}

C.1.2 SAC W/ ACTION REPETITION

- Action repetition length[4]: 3
- Rollout Length: {**4**, 8, 16, 33}

Other hyperparameters are kept the same as the optimal SAC ones.

[3]The target entropy used for automatically adjusting α is calculated as: $\sum_i[\ln(M_i - m_i) + \ln \Delta]$ where M_i/m_i are the maximium/minimum value of action dim i. Intuitively, the target distribution concentrates on a segment of length $(M_i - m_i)\Delta$ with a constant probability.

[4]Chosen to match the option interval K of HIDIO.

C.1.3 HIDIO

The hyperparameters of HIDIO were mostly heuristically chosen due to the hyperparameter search space being too large.

- Latent option $\mathbf{u}$ vector size: {**8**, 12}
- Policy/Q network hidden layers for π_ϕ : (128, 128, 128)
- Steps per option (K): 3
- π_ϕ has a fixed entropy coefficient α of 0.01. Target entropy min prob Δ for π_θ is 0.2.
- Discriminator network hidden layers: (64, 64)
- Replay buffer length per parallel actor: {50000, **200000**}
- Rollout Length: {**25**, 50, 100}

C.1.4 HIRO

- Steps per option: {**3**, 5, 8}
- Replay buffer size (total): {500000, **2000000**}
- Meta action space (actions are relative, *e.g.*, meta-action is `current_obs + action`): `(-np.ones(obs_space - 3D_goal_pos)*2, np.ones(obs_space - 3D_goal_pos)*2)`
- Policy stddev noise: {**0.1**, 0.3, 0.5}
- Number of gradient updates per training iteration: {100, **200**, 400}

C.1.5 HiPPO

For most hyperparameters, the search ranges chosen were derived after discussion with the first author of HiPPO.

- Learning rate: 3×10^{-4}
- Policy network hidden layers: (256, 256)
- Skill selection network hidden layers: {(32, 32), **(128, 64)**}
- Latent skill vector size: {5, **10**, 15}
- PPO clipping parameter: {0.05, **0.1**}
- Time commitment range: {(2, 5), **(3, 7)**}
- Policy training steps per epoch: {**25**, 50, 100}

C.2 SocialRobot

For all methods, we select the hyperparameters with the best area under the evaluation success curve (AUC) in the KickBall environment, and apply them to both KickBall and GoalTask. The shared hyperparameters are as follows (if applicable to the algorithm, and except when overridden by the respective algorithm's list of hyperparameters):

- Number of parallel actors/environments per rollout: 10
- Steps per episode: 100 (GoalTask), 200 (KickBall)
- Batch size: 1024
- Learning rate: 5×10^{-4} for all network modules
- Policy/Q network hidden layers: (256, 256, 256) with ReLU non-linearities
- Polyak averaging coefficient for target Q: 0.95
- Target Q update interval (training iterations): 1
- Training batches per iteration: 100

- Episodes per evaluation: 100
- Evaluation interval (training iterations): 100
- Initial environment steps for data collection before training: 100000

The training terminology here generally follows section C.1.

C.2.1 SAC

- Target entropy min prob Δ: $\{0.1, \mathbf{0.2}, 0.3\}$
- Replay buffer length per parallel actor: $\{20000, \mathbf{100000}\}$
- Rollout length: $\{12, 25, \mathbf{50}, 100\}$

C.2.2 SAC w/ Action Repetition

- Action repetition length[5]: 3
- Rollout Length: $\{4, \mathbf{8}, 16, 33\}$

Other hyperparameters are kept the same as the optimal SAC ones.

C.2.3 HIDIO

Due to the large hyperparameter search space, we only search over the option vector size and rollout length, and select everything else heuristically.

- Latent option $\mathbf{u}$ vector size: $\{\mathbf{4}, 6\}$
- Policy/Q network hidden layers for π_ϕ (128, 128, 128)
- Steps per option (K): 3
- π_ϕ has a fixed entropy coefficient α of 0.01. Target entropy min prob Δ for π_θ is 0.2.
- Discriminator network hidden layers: (32, 32)
- Replay buffer length per parallel actor: 20000
- Rollout Length: $\{50, \mathbf{100}\}$

C.2.4 HIRO

- Learning rate: 3×10^{-4}
- Steps per option: $\{\mathbf{3}, 5, 8\}$
- Replay buffer size (total): $\{\mathbf{500000}, 2000000\}$
- Meta action space (actions are relative, *e.g.*, meta-action is `current_obs + action`):
  - GoalTask: `(-np.ones(obs_space) * 2, np.ones(obs_space) * 2)`
  - KickBall: `(-np.ones(obs_space - goal_space) * 2, np.ones(obs_space - goal_space) * 2)` (because the goal position is given but will not change in the observation space)
- Policy stddev noise $\{\mathbf{0.1}, 0.3, 0.5\}$
- Number of gradient updates per training iteration: $\{100, 200, \mathbf{400}\}$

C.2.5 HiPPO

- Learning rate: 3×10^{-4}
- Policy network hidden layers: $\{\mathbf{(64, 64)}, (256, 256)\}$
- Skill selection network hidden layers: $\{\mathbf{(32, 32)}, (128, 64)\}$

[5]Chosen to match the option interval K of HIDIO.

- Latent skill vector size: {4, **8**}
- PPO clipping parameter: {**0.05**, 0.1}
- Time commitment range: {**(2, 5)**, (3, 7)}
- Policy training steps per epoch: {25, **50**, 100}