SHIRO: Soft Hierarchical Reinforcement Learning with Off-Policy Corrections

Anonymous Author(s)
Affiliation
Address
e-mail

Abstract:
Hierarchical Reinforcement Learning (HRL) algorithms have been demonstrated to perform well on high-dimensional decision making and robotic control tasks. However, because they solely optimize for rewards, the agent tends to search the same space redundantly. This problem reduces the speed of learning and achieved reward. In this work, we present an Off-Policy HRL algorithm that maximizes entropy for efficient exploration. The algorithm learns a temporally abstracted low-level policy and is able to explore broadly through the addition of entropy to the high-level. The novelty of this work is the theoretical motivation of adding entropy to the RL objective in the HRL setting. We support this with a proof showing that the entropy can be added to both levels if the Kullback-Leibler (KL) divergence between consecutive updates of the low-level policy is sufficiently small. We performed an ablative study to analyze the effects of entropy on hierarchy, in which adding entropy to high-level emerged as the most desirable configuration. Furthermore, a higher temperature in the low-level leads to Q-value overestimation and increases the stochasticity of the environment that the high-level operates on, making learning more challenging. Our method, SHIRO, surpasses state-of-the-art performance on a range of simulated robotic control benchmark tasks and requires minimal tuning.

Keywords: Off-Policy, Hierarchical, Reinforcement Learning, Maximum Entropy

1 Introduction

Deep Reinforcement Learning (RL) is a promising approach for robots to learn complex policies; e.g., manipulators [1], quadrupedal robots [2, 3] and humanoids [4, 5, 6]. Learning complex policies is made possible with the help of strong function approximators such as neural networks. This enables an agent to learn a policy in both a high dimensional state and action space [7]. However, Deep RL methods are well-suited for learning one atomic skill; e.g., a manipulator reaching an object or a quadruped walking upright. While very powerful, such methods struggle to learn compositional, long-horizon skills, such as utilizing locomotion for solving a maze. To solve such tasks, a better structure and stronger exploration strategies are required.

To this end, Hierarchical Reinforcement Learning (HRL) has been empirically shown to work well on these tasks by making use of the temporal abstraction generated from its hierarchical structure [8, 9, 10, 11]. In HRL, policies are stacked into multiple levels to construct a hierarchy, where the low-level agent receives information from the one(s) above it. This allows for the decomposition of a long-horizon task into smaller, more easily solvable problems. The theory of HRL has been well-studied over the years, but has lacked a generalized method that can train an agent efficiently in a complex, continuous space [12, 13, 14, 15, 16].

One potential solution is the state-of-the-art algorithm named Hierarchical Reinforcement Learning With Off-Policy Correction (HIRO) [8]. It proposes to stack two off-policy Deep RL agents and to relabel past experiences in the high-level policy’s replay buffer to correct the changes in the low-level policy. This allows training both levels simultaneously using an off-policy training method.
which speeds up the training. Accordingly, this method has been extended to new algorithms [11, 10, 17, 18, 19]. However, because the agent solely optimizes for reward, it over-commits to actions it believes are optimal, leading to poor exploration. This leads to inefficient training and slower convergence.

We present an off-policy hierarchical reinforcement learning algorithm (SHIRO: Soft Hierarchical Reinforcement Learning with Off-Policy Correction). We propose adding entropy maximization to the objective [20] of both levels of the agent to improve the training efficiency of HIRO. The addition of entropy in the RL objective has demonstrated an improvement of efficiency in exploration [21, 22, 23, 24, 25, 20, 26]. We extend this addition of entropy to the HRL setting, adding it to HIRO. Additionally, we justify the use of entropy maximization in the HRL context with a proof: a relatively small Kullback-Leibler (KL) divergence in the low-level agent over time guarantees that the low-level agent can be viewed as part of the environment. This allows us to train the high-level agent as if it was a single agent and applies the soft policy iteration theorem in SAC to our work. We empirically demonstrate more efficient learning with the addition of entropy to the RL objective. Our method produces superior results in data efficiency and training time to the base implementation of HIRO on HRL benchmark environments for robotic locomotion (AntMaze, AntFall, and AntPush). Finally, we conduct an ablative study, where we apply different combinations of entropy, and experiment with making the entropy temperature a learned parameter. The results of the ablative study are used to produce the best possible method.

## 2 Related Work

The problem of finding a suitable hierarchical structure to solve a long-horizon task has been widely explored in the RL community for many years [12, 13, 14, 15]. General structures are a tree-like structure where multiple policies are stacked like a tree and a tower-like structure where single policies are stacked on top of each other. Classic tree-like works train a high-level policy, given a set of hand-engineered low-level policies [27, 28, 29]. One popular framework in a tree-like structure is the Options framework [14, 30]. This algorithm learns to pick an option and follows the selected option policy until it meets its termination condition. We can view the high-level policy as a switch over options. However, it requires prior knowledge for designing options. To this end, the option-critic [31] proposes a method to train the high-level policy jointly. Recent work on the Options framework utilizes regularizers to learn multiple sub-policies [31, 32, 33].

Recent works focus on designing rewards to learn sub-policies more effectively. One choice is to learn both high-level and low-level policies from final task rewards end-to-end [34, 35], including the aforementioned option-critic frameworks [31, 32]. Another major stream is to provide auxiliary rewards to low-level policies to foster the learning [29, 36, 37, 38, 39], such as hand-engineered low-level reward based on domain knowledge [40, 36, 37, 38], mutual information for more efficient exploration [41, 39] and goal-conditioned rewards [12, 42, 43, 9, 44, 8, 11].

The tower-like structure, such as the FeUdal framework [12, 44] and HIRO algorithms [8], generally takes a form of goal-conditioned rewards, so that the high-level policy acts as a Manager/Planner to lead the low-level policy to work for/reach the sub-goal that the high-level policy provided. The benefit of these methods is that the representation is generic and does not require hand-engineering specific to each domain. Especially, HIRO [8] uses the state in its raw form to construct a parameterized reward and [11, 10] extend the sub-goal to latent spaces. This goal-conditioned framework is also extended to the real robotic domain [17, 45] and extensively explored outside of HRL [46, 47, 42, 43, 48]. Problems of these algorithms are also targeted as in [49, 50, 51, 52], indicating that there are numerous avenues for improvement.

Furthermore, while very powerful, HIRO searches the same states redundantly, leading to inefficiency. Our algorithm extends the work to maximize entropy in addition to the original RL objective, that is shown to work well in other works [21, 22, 23, 24, 25, 20, 26], including finding diverse goals to speed up learning [53]. It is also applied in HRL [54], but not in the goal-conditioned setting. [19] is the closest to our work where they use SAC on both levels, but they don’t explain on what condition entropy works well. While maximizing entropy is effective, too high entropy leads to poor performance. Regarding the problems of sub-goals, [18] added a constraint to avoid from generating unreachable sub-goals, and [53] minimizes changes in the high-level policy which facilitates learning.
Figure 1: The left figure is a schematic of how the agent interacts with the environment. The high-level policy generates a sub-goal to guide the low-level policy in its direction as shown in the center figure. The right figure is the depiction of the high-level policy generating sub-goals at time $t$ in AntMaze (from Figure 2). The entropy maximization helps the policies to diversify their sub-goals/actions for better exploration, allowing to get out of the local optima, faster convergence, and improved training of the low-level policy.

Unlike other methods, our work adds entropy to either or both levels and explains on what condition adding entropy works and how to best utilize entropy.

3 Preliminaries

3.1 Goal Conditioned Reinforcement Learning

Goal-Conditioned Reinforcement Learning (RL) extends the definition of the RL problem by conditioning its policy and reward function on goals. To define formally, a Markov Decision Process (MDP) is augmented with a set of goals $G$, and is called a Universal MDP (UMDP). A UMDP is a tuple containing $(S, G, A, P_T, R)$, where $S$ is the state space, $A$ is the action space, $P_T : S \times A \times S \rightarrow [0,1]$ is the transition probability and $R : S \times G \times A \rightarrow [r_{min},r_{max}]$ is the reward function mapping state, goal and action pair to a scalar value reward between $r_{min}$ and $r_{max}$. The reward function is conditioned on a goal to represent how well the agent is approaching the goal. To approach each goal in a different manner, the agent’s policy $\pi_t = \pi(a_t|s_t, g)$ is augmented to condition on the goal. At the start of every episode, a goal $g$ is sampled from a set of goals $G$ by the distribution $P_g(g)$. At every time step $t$, the environment generates state $s_t \in S$. Then, the agent takes action $a_t \in A$ sampled from its policy $\pi(a_t|s_t, g)$, transitions to the next state $s_{t+1}$ with probability $P_T(s_{t+1}|s_t, a_t)$, and receives a reward $R_t = R(s_t, a_t, g)$. The probability distribution over the samples generated by the policy $\pi$ is denoted as $\rho^\pi$. The objective is to maximize the expected reward, or $\mathbb{E}_{(s_t,a_t) \sim \rho^\pi, \ g \sim P_g} [\sum_{t=0}^{\infty} R(s_t, a_t, g)]$.

3.2 Goal Conditioned Hierarchical Reinforcement Learning

HRL is an RL approach that leverages the hierarchical structure in an agent’s model to decompose the original problem into sub-problems. This allows the algorithm to learn within the space of each sub-problem, and to learn a policy that can be shared across the sub-problems. Hierarchical Reinforcement learning with Off-policy correction (HIRO) [8] is a potential solution to this problem that makes use of goal-conditioned policies. This algorithm successfully accomplishes continuous complex tasks and is widely used [10, 55, 11]. Our work is based on their work, and is depicted in Figure 1. In this algorithm, the agent consists of a high-level policy $\pi^h(g_t|s_t, g)$ and a low-level policy $\pi^l(a_t|s_t, g_t)$, where $g_t$ is a sub-goal sampled from the high-level policy. At every $c$ steps, the high-level policy computes a sub-goal for the low-level policy given the current state $s_t$ and final goal $g$. The low-level policy is conditioned on the current state $s_t$ and the sub-goal $g_t$ to generate an action $a_t$. For the rest of the steps, the sub-goal is advanced by the goal transition model $h(s_t, g_t, s_{t+1}) = s_t + g_t - s_{t+1}$ based on the current and next state. The intrinsic reward is defined as $r(s_t, g_t, s_{t+1}) = -||g_t - g_t - s_{t+1}||^2$ to reward the low-level policy, while the environment reward is for the high-level policy. To train the policies, the experiences are stored into replay buffers.
we introduce another function called the Abstracted Transition Function $T$ where we define an Abstracted Reward Function $R$. The policy is then trained to maximize the Q-value at each state $\pi(s_t, a_t) = \arg\max_{a \in A} Q^\pi(s_t, a)$. However, the standard formulation faces two major challenges; high sample complexity and brittle convergence. To this end, Soft Actor-Critic (SAC) [20] proposes to alter the original objective to include entropy maximization, i.e., $J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho^\pi} [R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))]$, where $\alpha$ is a temperature parameter that tunes the stochasticity of the optimal policy and $\mathcal{H}$ represents the entropy of the policy. The Q-function and value function can be computed iteratively by:

$$Q^\pi(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P_T} [V^\pi(s_{t+1})]$$

$$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi} [Q^\pi(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$$

The last term adds a larger value to the original Q function if the probability of taking action $a$ is small so that it explores actions that are less likely to occur. By adding entropy, it improves exploration, allowing for searching sub-optimal actions that may potentially lead to better rewards.

### 4 Soft Hierarchical Reinforcement Learning

While HIRO is very effective, it searches the same areas redundantly (see Section 5), which contributes to data inefficiency, slowing down learning. In our algorithm, we propose to add entropy to the high-level policy to improve exploration. Furthermore, we also show how it can be best added to both levels. In Maximum Entropy RL, e.g., the Soft Actor Critic algorithm, the objective is to maximize the expectation of accumulated rewards and the policy entropy for a single agent. Similarly, in SHIRO, we also maximize entropy as follows.

$$J(h) = \sum_{i \in T_c} \mathbb{E}_{(s_t, g_t) \sim \rho^h} \left[R^{abs}(s_t, g_t) + \alpha \mathcal{H}(\pi(\cdot|s_t, g))\right]$$

where we define an Abstracted Reward Function $R^{abs}(s_t, g_t) = \sum_{t'=t}^{t+c} R(s_{t'}, \pi'(a_{t'}|s_{t'}, g_t))$. In the case of goal-conditioned HRL, the high-level agent provides a new sub-goal every $c$ steps.

By introducing such a function, the objective can become analogous to the standard reinforcement learning objective if we assume that the low-level policy is a part of the environment. Given this assumption, in order to transform the HRL problem into one that can be applied to standard RL, we introduce another function called the Abstracted Transition Function $P_T^{abs}(s_{t+c}|s_t, g_t)$; i.e., a transition function from a state $s_t$ to a state $s_{t+c}$, given a sub-goal $g_t$. With the abstraction, the probability of a trajectory $\tau$ can be rewritten as follows, where the original probability for the single policy is given as a multiplication of probabilities of state and action sequences along the trajectory, $p(\tau) = p(s_0) \prod_{t \in T_c} [\pi^h(g_t|s_t, g)] P_T^{abs}(s_{t+c}|s_t, g_t)$.

Given that the low-level policy gets updated, which induces a certain degree of stochasticity, we can view this Abstracted Transition Function as a representation of the stochastic environment. This is exactly the same problem as in the standard Reinforcement Learning objective for the high-level policy, as long as the Abstracted Transition function does not change drastically. Formally, we can bound the change in the Abstracted Transition function by the following theorem:

**Theorem 1.** Let $\pi^h_0(a_t|s_t, g_t)$ be the low-level policy before the parameter update and $\pi^h_\theta(a_t|s_t, g_t)$ be the updated low-level policy after $c$ steps. If two policies are close, i.e., $|\pi^h_\theta(a_t|s_t, g_t) - \pi^h_0(a_t|s_t, g_t)| \leq \epsilon$ for all $s_t$, then the change in the abstracted transition functions is bounded by a small value $|P_T^{abs}(s_{t+c}|s_t, g_t) - P_T^{abs}(s_{t+c}|s_t, g_t)| \leq 2\epsilon c$, where $\epsilon \in [0, 1]$.

See the proof in Appendix A.
4.1 Addition of Entropy to Each Policy

**High-Level Policy:** If the changes in the parameters are small, and if we can assume the low-level policy as a part of the environment, we can view the HRL problem in the same way as in the original RL problem, shown in Equation (3). Then, we can derive the exact same theorems as in SAC [20] for the high-level policy. For a deterministic high-level, we can apply the theorems of standard policy improvement and iteration. The addition of entropy to high-level is desirable for improved hierarchical exploration because it will generate more diverse sub-goals that facilitate learning of the low-level, leading to a better performing low-level agent to reach those broad sub-goals.

**Low-Level Policy:** Simultaneously (or by itself), we can add entropy to the low-level policy according to Theorem 1. To make the total variation divergence of policies between the parameter update small, we can use Pinsker’s inequality, i.e., $|\pi_{\theta'}(a_t|s_t, g_t) - \pi_{\theta}(a_t|s_t, g_t)| \leq \sqrt{\frac{1}{2}KL(\pi_{\theta'}(a_t|s_t, g_t)||\pi_{\theta}(a_t|s_t, g_t))}$. This means that if the KL divergence between parameter updates is kept low, we can bound the changes in the Abstract Transition function and allow the low-level policy to be part of the environment. In this paper, we will not present a constraint to minimize the KL divergence during training; instead, we show here why high-level and both-levels agents can perform better than others. However, in Section 5, we will verify that the KL divergence is sufficiently small during training. Moreover, the temperature parameter should not be too large to avoid the second term in Equation (2) to be large, which induces an incorrect estimation of the value function. This induces overly high values on actions that are unlikely to be taken, messing up the estimation of the low-level policy.

4.2 Learning Temperature Parameter

Given that our proof allows to view the high-level agent as a single SAC agent, this motivates the use of learned entropy on the high-level. We apply the learned temperature method of [58] into the high-level and low-level agent. The objective for learning entropy temperature is given by $\arg\min_\alpha \mathbb{E}_{a_t \sim \pi_t} \left[ -\alpha \log \pi_t^*(a_t|s_t, g_t) - \alpha \mathcal{H} \right]$, where $\mathcal{H}$ is the minimum desired entropy, and $\pi_t^*$ is the policy at timestep $t$. The expression is a measure of the difference between the current policy’s entropy and the minimum desired entropy. A lower value of $\alpha$ will minimize and scale down this difference. With respect to the high-level, we assign a high temperature at the beginning of training to speed up learning, but allowing the temperature to decrease once the agent has learned. Furthermore, this is sufficient as the low-level agent will have been sufficiently trained on diverse data initially; lowering entropy does not pose any major concern. Similarly, we can also learn the temperature parameter $\alpha$ of the low-level agent, in which we desire a low entropy that won’t increase significantly.

5 Experiments

We conducted two experiments to demonstrate the performance of our algorithms. The first experiment compares our best method to HIRO in benchmark environments. The second experiment is an ablative analysis of our algorithm. In this study, entropy is added to either or both of the levels, as well as with learning and not learning the temperature to analyze the behavior of the algorithm. These experiments demonstrate the power of the maximum entropy hierarchy with a simulated robot in MuJoCo.
Concretely, the agent is modeled as in Figure 1. The structures and the hyperparameters of the policies and Q-functions in the deterministic case are detailed in the original paper [8]. Most importantly, the entropy is defined and added to the original objective as in Equation (3). The entropy is added to either or both of the policies, and they are compared against HIRO, as in Figure 3. When we maximize entropy on either layer, we use the parameters according to Haarnoja’s ([20]) SAC. We call each of the variants, SHIRO HL (high-level entropy), SHIRO LL (low-level entropy), and BL (entropy at both levels) and used a temperature of 1.0 for high-level policy and 0.1 for the low-level policy across all environments. Other hyperparameters were carried over from SAC. We also used the same value as the initial temperatures for agents with temperature learning (SHIRO HL-Learned, SHIRO LL-Learned, SHIRO BL-Learned). All of the implementations is built on top of PFRL in PyTorch [59].

5.1 Comparative Study

In this study, we compared our best performing method, SHIRO-HL Learned, against HIRO [8] to demonstrate the effectiveness of entropy maximization formulation. HIRO is known to perform better than other HRL methods such as FeUdalNet[44] and Option-Critic[31] on the same tasks [8]. Furthermore, we also compared our methods to the vanilla SAC (no hierarchy).

In our research, we used the same environment that was used in the original paper of HIRO [8] to compare against their algorithm. We tested our algorithm on three of their environments: AntMaze, AntPush, and AntFall, as shown in Figure 2. All three environments require robotic locomotion and are standard benchmarks of HRL. As presented in Figure 3, in all environments, our method starts succeeding two to four times faster than the standard HIRO. Although the success rate increases steadily for AntMaze, it fluctuates for AntPush and AntFall, but gradually increases after the drop. This is thought of as a result of exploration for a better policy: Entropy maximization fosters exploration in new regions. The effect of broad exploration is shown in Figures 4a and 4b. The figure represents the scatter and contour plot of the final positions achieved by an agent in each environment. From the figure, we can observe that HIRO agent is failing to explore the areas around the goal but repetitively searches around the starting position before eventually learning locomotion to achieve the goal. In comparison to HIRO, qualitatively, our method is being able to reach goals more broadly.

5.2 Ablative Study

In Figure 5, we display the results of our method SHIRO with different configurations that we tested. There are a few different ways to add entropy to a hierarchical agent, so we performed an ablative study to find out the best configuration.

Adding Entropy to Each Layer. We compare agents with entropy on different levels (SHIRO HL/LL/BL), as well as SAC. Each run result consists of three experiments using the same randomly selected seeds for each ablation. Figure 5a shows our results. We found that while all ablations perform better than the standard HIRO (besides SAC, which failed to complete the task), some are better than others. Adding entropy to the high-level controller yielded results that were consistently better across the board. Adding entropy to the low-level controller didn’t perform as well, starting to succeed faster than regular HIRO, but took longer to reach the maximum success rate (Details in Section 5.3). Finally, adding entropy to both levels performed at around the same level as adding it to just the high-level controller.
Figure 4: Top: Analysis of the final positions achieved by an agent in a given environment. The left plot is simply a scatterplot of the final positions of an agent on the environment specified underneath. The second plot displays a contour map showing where the agent ends up on average across the entire experiment. We find that across all environments tested on, SHIRO HL-Learned explores the environment more thoroughly than HIRO. Bottom: X-Y Sub-goal visualization of the high-level policy.

Figure 5: Ablative Study

We find that the learned temperature parameter plays a role in the performance of our ablations, as in Figure 5b. If the temperature is high, the agent is incentivized to select actions (sub-goals in the case of the high-level agent) that provide more entropy. With respect to the high-level, while this is good when the agent is starting to learn, we want the agent to start acting deterministically as its success rate reaches its peak. We can observe this behavior in one of the figures in Appendix C, where the high-level temperature generally decreases with time according to the performance of the agent.

High-Level vs. Both-Levels Since both SHIRO HL-Learned and BL-Learned perform well, it is necessary to compare them in order to find the best option, shown in Figure 5c. We find that SHIRO HL-Learned performs marginally better than SHIRO BL-Learned. Thus, we consider SHIRO HL-Learned to be our best method produced from the ablative study, as well as the fact that SHIRO HL-Learned does not require any temperature parameter tuning on the low-level. However, we suggest that both methods are viable, and one or the other can be chosen according to preference.
5.3 Analysis

In our experiments, the KL-divergence in the low-level policy is sufficiently low in our ablations, which matches the theory presented in Theorem 1 that can be applied to HIRO with and without high-level entropy. In general, the KL divergence is below 1.0 (a plot can be found in Appendix C). However, even though SHIRO with a low-level temperature of 1.0 had the lowest KL divergence, it fails to learn because the high temperature parameter induces Q-value overestimation. As the temperature increases, the second term in Equation (2) becomes large that overestimates the actions that are unlikely to be taken. It also creates a too stochastic environment for the high-level agent to solve the main task.

We also observe that the addition of entropy to the high-level agent significantly improves the learning of the low-level agent, in turn, improving the high-level agent by making the environment more "helpful" to solve the main task, as well as being able to broadly explore the environment. The more diverse sub-goals provided to the low-level agent increase the loss (see Appendix C) of that agent’s Q-functions because the Q-functions must initially learn from more diverse data (this difference in diversity can be seen in Figure 4c). This allows for larger updates (from gradient descent) that enable the Q-function to learn from underrepresented goals. With a more robust Q-function that has received more meaningful learning signals, we also observe an improvement in the low-level policy loss (details in Appendix C), where the policy directly optimizes from the Q-function. In turn, the low-level agent learns locomotion quickly.

Another observation from our study is that high-level entropy maximization improves slightly “worse” low-level agents. When the high-level agent is kept deterministic, selecting a low-level policy as SAC (which has empirically shown to outperform TD3) demonstrates better performance than a TD3 low-level policy. However, when the high-level policy maximizes entropy, the performance of different low-level agents significantly improves but are similar to one another, suggesting that the diversity of data significantly improves worse low-level agents. Graphs of these observations are located in Appendix C.

6 Conclusion

In this work, we proposed an effective solution to the problem of redundant sub-goals for robotic learning in goal-conditioned hierarchical reinforcement learning. We achieved this through maximum entropy goal-conditioned HRL, in which we theoretically showed that that we can view our method as a single agent if the KL divergence is small between the low-level policy updates. Our theoretical findings motivate the learning of entropy temperature, namely on the high-level. We performed an ablative study to understand the effects of entropy on both levels, where high-level learned entropy had significant impact on the efficiency. Concretely, the diversity of sub-goals provided to the low-level agent improved the redundancy and reachability of goals, allowing for a high-level agent to learn even faster. Regarding future work, we are interested in applying our findings on a real robot platform, as well as exploration of the importance of sub-goal representations.

References


11