HRL2E: Hierarchical Reinforcement Learning with Low-level Ensemble

You Qin, Zhi Wang* , Chunlin Chen
Department of Control and Systems Engineering
Nanjing University
Nanjing, China
youqin@smail.nju.edu.cn, {zhiwang, clchen}@nju.edu.cn

***Abstract*—Goal-conditioned hierarchical reinforcement learning (HRL) is a promising approach to solve challenging tasks with sparse rewards and long horizons. However, it suffers from the non-stationary problem due to the updating and unstable low level. To stabilize the low level more quickly and accelerate the non-stationary stage, we propose a novel HRL method: Hierarchical Reinforcement Learning with Low-level Ensemble (HRL2E). In HRL2E, the high level generates goals as high-level actions based on current states. Then the low level made up of several homogeneous policies attempts to complete these goals within a specific timestep budget. The improvement of our approach to the general goal-conditioned HRL algorithms can be summarized in two aspects. First, we estimate the target value function with the ensemble, stabilizing the training process. Second, we propose the Gates module composed of several scoring machines to score each low-level policy and judge which one has the most success potential to execute a specific goal. We adopt Twin Delayed Deep Deterministic Policy Gradient (TD3) in each level. Experimental comparison between our method and state-of-the-art goal-conditioned HRL methods on challenging continuous control tasks in MuJoCo domains shows our method can significantly accelerate training.**

I. Introduction

Deep Reinforcement Learning (RL), in which the agent learns knowledge from trial and error with the environment, has made significant progress in many domains like games [1] and robot control [2]. Nevertheless, it remains difficult for RL to solve tasks with long time horizons and sparse rewards. These complex tasks can generally be decomposed into smaller ones with relatively short horizons and dense rewards. So we naturally think of Hierarchical RL (HRL), a hierarchical algorithm with different temporal abstraction degrees among levels. The efficiency of HRL can be attributed to more efficient exploration than general RL [3]. In the end-to-end goal-conditioned paradigm, the high-level policy continuously generates goals at a greater temporal granularity. In comparison, the low-level policy attempts to achieve them within a specific timestep budget at a finer temporal granularity.

However, there is a severe non-stationary problem of its training process, i.e., the low level cannot completely and stably execute each goal given by the high level. Hence, the state transition function of the high level is not stable. Of course, we can adopt on-policy algorithms in each hierarchy to avert the non-stationary problem, but it is not sample-efficient. So we do not consider that in our work. The training process of end-to-end HRL algorithms can be roughly divided into two stages [4]: (1) non-stationary stage; (2) near-stationary stage. In the first stage, the low-level policy varies widely with the update of the networks. However, the high level needs to explore the environment with the unstable low-level policy. It is inefficient but ineluctable for end-to-end HRL. During the second stage, the low level is stable and can basically complete the high-level actions. Hence, the high-level policy can use stable and efficient low-level policy to explore the environment. Therefore accelerating the first stage and stabilizing the low level earlier can promote remarkable efficiency to end-to-end HRL algorithms.

Inspired by the social structure, a multilayered structure with more units in lower levels, we propose a novel approach named Hierarchical Reinforcement Learning with Low-level Ensemble (HRL2E). The main idea of our proposal is to use an ensemble of K homogeneous low-level policies to shorten the first stage of end-to-end HRL. The reason why we choose ensemble is three-fold. First, we use ensemble to average the learned Q-value estimates, which can stabilize the training process and promote the low-level performance by reducing approximation error variance in the target values. This technique is firstly used in ensemble DQN [5] with the theoretical proof. Second, We alleviate the sample efficiency problem with the ensemble. For common sampling methods from experience replay buffer, the policy could not select the most informative experiences and learn all knowledge included in the replay buffer, which leads to large samples waste and more interaction data. Therefore, we use ensemble to improve the effective utilization of samples by respectively training K low-level policies with experiences randomly sampled from the experience replay buffer. Hence, the ensemble could learn more knowledge from the replay buffer. Third, we can reduce the training difficulty by scheduling low-level policies correctly. In those complex environments with large continuous state space, it could be problematic for low-level policy to find beneficial actions in the large state-goal united space. So we think of the ensemble to enhance the representation ability of the low level. Therefore, by scheduling low-level policies to execute goals with the most success potential, we can implicitly divide the large continuous state-goal united

*Corresponding author

space into K sub-spaces and reduce the training difficulty of the low level. To our best knowledge, this is the first time that ensemble is used in HRL.

To schedule low-level policies properly, we propose a structure named Gates to predict the success potential of each low-level policy. With this, we can pick out the most proper low-level policy to execute the specific goal at a certain state. Therefore, the high level could do deeper exploration with the effective low level and sample more informative high-level experiences. We judge how close the agent is to the goal after the timestep budget and then score the related low-level policy.

We compare our method with state-of-the-art end-to-end goal-conditioned HRL algorithms in MuJoCo domain [6]. The results present that our method outperforms existing algorithms.

We summarize the main contributions of this paper as follows:

- We leverage the ensemble in HRL for the first time, which accelerates the non-stationary stage.
- We propose the Gates module to evaluate the success potential of each low-level policy and decide whether to schedule it.
- Experiments demonstrate the remarkable effect of our method in MuJoCo environments

II. Related Work

HRL algorithms. Recently, many various HRL algorithms have occurred. Most of them can be roughly divided into two categories, feudal framework [2], [7] and options framework [8]. The former generally comes in goal-conditioned end-to-end form [9]–[13]. In the options framework, we often need to pre-train options or skills [14], [15]. While there are still end-to-end option methods to automatically learn skills [16]–[18]. In this work, we focus on the feudal framework. There are two stages in the training of goal-conditioned end-to-end HRL algorithms because of the non-stationary problem [4]. Some works like [7], [14], [19] adopt an on-policy RL algorithm in each level to avert the non-stationary problem, but this leads to the loss of sample efficiency. In our algorithm, we use the off-policy algorithm TD3 [20] in each level and utilize ensemble to accelerate the non-stationary stage.

Ensemble in RL. To our best knowledge, no direct combination work between ensemble and HRL has been done yet. Most works just concentrate on ensemble and flat RL algorithms [21]–[23]. [5] proposes Averaged DQN to decrease Q-variance with k set of historical Q-network parameters and Ensemble DQN with k Q-networks. Bootstrapped DQN with multi-heads (Q-networks) is proposed to increase deep exploration [22]. Sunrise method [23] leverages ensemble to improve exploration and decrease error propagation of Q-estimation, which exhibits much better performance than general RL algorithms. In our method, we consider leveraging ensemble to improve the low-level performance and accelerate the non-stationary stage.

Intrinsic reward in HRL. In RL, reward design directly determines the behaviors of the agent. In HRL, the high level accepts the rewards from the environment guiding the agent to complete the whole task, while the low level accepts intrinsic rewards guiding to the high-level actions. Therefore, proper low-level reward design can improve the reachability of goals and alleviate the trade-off between temporal abstraction and goal reachability [24]. The reachability issue could be worse if we adopt sparse low-level rewards. Fortunately, the high-level action space is a latent space mapped from the state space, so we can utilize dense intrinsic reward. [7] integrates direction-heuristic reward and environment reward as the low-level reward. [14] sets low-level reward based on the advantage function of the high-level policy. In this work, we adopt negative Euclidean distance as intrinsic rewards for the low level. To train the Gates module, we use the negative Euclidean distance between the end state of a goal trajectory segment and the goal state to score low-level policies.

Scheduling in ensemble. How to promote exploration in RL is a crucial and long-standing problem. Leveraging ensemble in RL has shown a significant effect to increase efficient exploration. The exploration strategy based on upper-confidence bounds (UCB) is often adopted to stimulate the agent to explore uncertain states [21], [23], [25], [26]. In bootstrap DQN [22], a specific deep Q-network is randomly selected and used with the greedy exploration strategy. [27] uses self-supervised Mixture-of-Experts to guide exploration according to estimation uncertainty generated directly by particular networks [28]. We use Gates module made up of several gate networks to score and schedule different low-level policies in our work.

III. Preliminaries

A. Reinforcement Learning

RL algorithms are generally based on Markov Decision Processes (MDP) [29], defined as a tuple: $(\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma)$, where $\mathcal{S}$ represents a state space, $\mathcal{A}$ is a action space, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a state transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function, $\gamma \in [0, 1]$ is the discount factor of future rewards. The RL agent learns to maximize the discount rewards sum $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t (s_t, a_t)]$ according to a certain policy π.

TD3 algorithm is a state-of-the-art off-policy RL algorithm, which alleviates the overestimation problem with target networks, double critics and target policy smoothing regularization. Therefore the target of the critic is defined as:

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \pi_{\phi'}(s) + clip(\epsilon, -d, d)) \tag{1}$$

,where $\epsilon \sim \mathcal{N}(0, \sigma)$ is the disturbance noise with policy to make the learned value function smoother, and d is a noise clipping threshold. Then the value estimate learned is with respect to a noisy policy defined by the parameter σ. We update critic parameters according to:

$$\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2 \tag{2}$$

The Actor is updated by the deterministic policy gradient:

$$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)\Big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s) \tag{3}$$

The target networks are updated at larger step intervals according to:

$$\begin{aligned} \theta'_i &\leftarrow \tau\theta_i + (1-\tau)\theta'_i \\ \phi' &\leftarrow \tau\phi + (1-\tau)\phi', \end{aligned} \tag{4}$$

where τ denotes the smoothness to adjust the degree of soft update.

B. Hierarchical Reinforcement Learning

HRL leverages multiple layers to solve long-horizon and sparse-reward RL problems. Considering a goal-conditioned HRL framework with two layers, we use π^h to denote the high-level policy and π^l to denote the low-level policy. The high-level action $g_t \sim \pi^h(s_t)$ means a goal to be achieved by the low level within c timesteps and $g_t \in \mathcal{G}$, where $\mathcal{G}$ denotes the goal space. The goal transition is completed according to the high-level policy $g_{t+1} = \pi^h(s_{t+1})$ when t mod $c = 0$, otherwise $g_{t+1} = h(s_t, g_t, s_{t+1})$, i.e.,

$$h(s_t, g_t, s_{t+1}) = f(s_t) + g_t - f(s_{t+1}) \tag{5}$$

where $f : \mathcal{S} \to \mathcal{G}$ is a goal mapping function from the state space to the goal space. With the observation of the current state s_t and goal g_t, the low level generates primitive actions $a_t \sim \pi^l(s_t, g_t)$ sequentially to interact with the environment. The environment will feed back reward signal $r(s_t, a_t)$ for each action a_t. Then the reward function for high-level actions is defined as the sum of the c rewards within the timestep budget:

$$r^h = \sum_{i=0}^{c-1} r(s_{t+i}, a_{t+i}) \tag{6}$$

The high level policy is to maximize the expected discounted return of environment:

$$J(\pi^h) = \mathbb{E}_{\pi^h}\left[\sum_{t=0}^{\infty} \gamma^t r_t^h\,(s_t, a_t)\right] \tag{7}$$

The low level accepts intrinsic reward that guides the agent to the goal state set by the high level. We define it as the negative Euclidean distance between the current state to the goal state:

$$r^l(s_t, g_t, s_{t+1}) = -||g_t - f(s_{t+1} - s_t)||_2 \tag{8}$$

For each high-level action, the goal of the low level is to find a policy to maximize the expected intrinsic return within c steps:

$$J(\pi^l) = \mathbb{E}_{\pi^l}\left[\sum_{t=0}^{c} \gamma^t r_t^l\,(s_t, g_t, s_{t+1})\right] \tag{9}$$

IV. METHODOLOGY

In this section, we present our method for accelerating the non-stationary stage and decreasing the training difficulty of the low level. First, we introduce the training process of high level and how to do the off-policy correction with the ensemble. Second, we present how to schedule low-level policies with the Gates module and leverage ensemble to reduce the approximation error variance in target values [5]. Lastly, we introduce the components of the Gates module and how it works. The overall framework of our method is presented as Fig. 1, and pseudocode as Alg. 1.

Algorithm 1 HRL2E

```
Input: number of ensemble components $K$, batch size $N, N'$,
    standard deviation of exploration noise $\sigma_h, \sigma_l$, noise clipping threshold $d$
Initialize: Initialize high-level critic and actor networks:
    $Q^h_{\theta_1}, Q^h_{\theta_2}, \pi^h_\phi$ and copy weights to target networks:
    $Q^h_{\theta'_1}, Q^h_{\theta'_2}, \pi^h_{\phi'}$, low-level: $\{Q_{k,\vartheta_1}, Q_{k,\vartheta_2}, \pi_{k,\varphi}\}_{k=1}^K$ and
    copy weights to target networks $\{Q_{k,\vartheta'_1}, Q_{k,\vartheta'_2}, \pi_{k,\varphi'}\}_{k=1}^K$,
    $Gates$: $\{G_{\psi_k}\}_{k=1}^K$
1:  for each iteration do
2:    Observe environment state $s_t$,
      goal $g_t \sim \pi^h(s_t) + \epsilon$, $\epsilon \sim clip(\mathcal{N}(0, \sigma_h), -d, d)$
3:    Set $s_{seq} \leftarrow [s_t]$, $a_{seq} \leftarrow [\ ]$
4:    for each timestep $t$ do
5:      Generate scheduling vector: $\boldsymbol{m}_t = Gates(s_t, g_t)$,
        adopted low policy: $\pi^l_{k,\varphi}$, where $k = argmax(\boldsymbol{m}_t)$
6:      Select action with exploration noise:
        $a_t \sim \pi^l_{k,\varphi}(s_t, g_t) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$
7:      Observe reward $r_t$, next state $s_{t+1}$, calculate intrinsic
        reward $r^l$
8:      $s_{seq}.append(s_{t+1})$, $a_{seq}.append(a_t)$
9:      if $t$ mod $c$ then
10:       Select goal with exploration noise:
          $g_{t+1} \sim \pi^h_\phi(s_{t+1}) + \epsilon$, $\epsilon \sim clip(\mathcal{N}(0, \sigma_l), -d, d)$
11:       Store transition: $(s_{t-c+1}, g_{t-c+1}, r^h, s_{t+1}, s_{seq}, a_{seq})$
          in $\mathcal{B}^h$, where $r^h = \sum_{j=t-c+1}^{t} r_j$
12:       Calculate $score_k$ as (14)
13:       Store sample: $(s_{t-c+1}, g_{t-c+1}, score_k)$ in $\mathcal{B}^G_k$
14:       Set $s_{seq} \leftarrow [s_{t+1}]$, $a_{seq} \leftarrow [\ ]$
15:     else
16:       Goal transition: $g_{t+1} = h(s_t, g_t, s_{t+1})$
17:     end if
18:     Store transition $(s_t, g_t, a_t, r^l, s_{t+1}, g_{t+1})$ in $\mathcal{B}^l$
19:   end for
20:   for each update step do
21:     Sample mini-batch $B$ of $N$ transitions
        $(s, g, R, s', s_{seq}, a_{seq})$ from $\mathcal{B}^h$
22:     Do off-policy correction as (11) and get new batch
        $B'$
23:     update twin $Q^h_{\theta_i}$, $\pi^h_{\phi_i}$ and target twin networks $Q^h_{\theta'_i}$,
        $\pi^h_{\phi'_i}$ as TD3 on mini-batch $B'$
24:     for $k$ from 1 to $K$ do
25:       Sample mini-batch of $N$ transitions
          $(s, g, a, r, s', g')$ from $\mathcal{B}^l$, and $N'$ samples
          $(s, g, score_k)$ from $\mathcal{B}^G_k$
26:       update twin $Q^l_{k,\vartheta_i}$, and $\pi^l_{k,\varphi_i}$; target twin networks
          $Q^l_{k,\vartheta'_i}$, and $\pi^l_{k,\varphi'_i}$ as TD3; update $G_{\psi_k}$ as (15)
27:     end for
28:   end for
29: end for
```

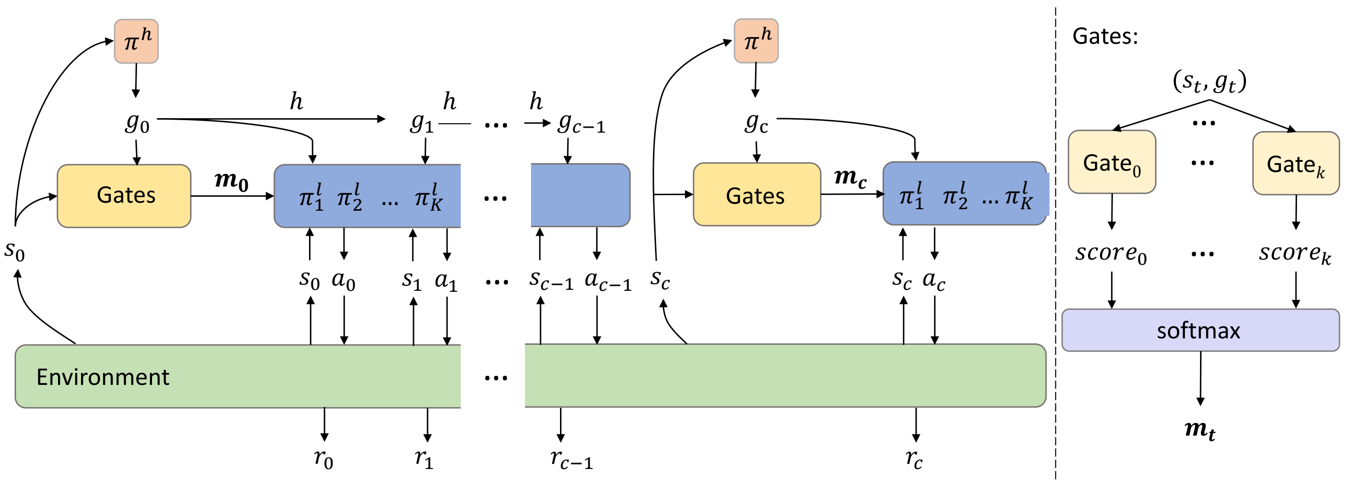


Fig. 1. Left part: overall of HRL2E. The high-level policy receives a state from the environment and generates a goal. According to the current state and goal, the Gates module generates a scheduling vector to select a low-level policy from K low-level policies to execute the goal within c steps. We adopt the same low-level policy during the goal's time budget and use h-function to do goal transition. After c steps, we acquire a high-level transition and c low-level transitions in related replay buffers. Right part: the Gates module. We train K networks to score each low-level policy and generate a scheduling vector $\boldsymbol{m_t}$.

A. *High-level Part*

In our method, we use the off-policy algorithm TD3 for each level. Because of the non-stationary problem, we need to do off-policy correction for high-level transitions $(s_t, g_t, R, s_{t+c}, s_{t+1}, s_{seq}, a_{seq})$, where s_{seq}, a_{seq} is state sequence and action sequence in the trajectory segment of the goal g_t. In HIRO algorithm [9], the off-policy correction is to maximize the log-likelihood of the sampled segment $\log \pi^h(a_{t:t+c-1}|s_{t:t+c-1}, \hat{g}_{t:t+c-1})$, given the current low-level policy. To maximize that in practice, we can approximate the goal after correction as:

$$\hat{g}_t^* = \arg\min_{\hat{g}_t} \frac{1}{2} \sum_{i=t}^{t+c-1} ||a_i - \pi^l(s_i, \hat{g}_i)||_2^2, \tag{10}$$

where $\hat{g}_t$ named candidate goal is sampled from a Gaussian distribution centered at $s_{t+c} - s_t$, including original goal g_t and center of the Gaussian distribution. There are several low-level policies in our method, and we need to select one of them to do the off-policy correction. Since low-level policies will all be gradually stable and able to achieve goals, we consider using a specific policy continuously in off-policy correction and the experiment results show that works. Consequently, the corrected goal can be approximated as:

$$\hat{g}_t^* = \arg\min_{\hat{g}_t} \frac{1}{2} \sum_{i=t}^{t+c-1} ||a_i - \pi_u^l(s_i, \hat{g}_i)||_2^2, \tag{11}$$

where $u \in [1, K]$ is a fixed hyperparameter. Then we can train the high-level policy with TD3 algorithm. Similar to Eq.2 and Eq.3, we just need to replace action a with corrected high-level action $\hat{g}$.

B. *Low-level Part*

There are K randomly initialized low-level policies in our method. As presented in Fig. 1, once the high level generates a new goal, we judge which low policy is more adept at this goal and then schedule it to execute this goal within a fixed timestep budget c. We adopt dynamic goal scheme in which we need to do goal transition after per primitive action, i.e., $g_{t+1} = h(s_t, g_t, s_{t+1})$. We do not use the scheduling module to reassign a new low-level policy for the goal generated by the goal transition function h rather than the high level. Similar to the bootstrap DQN [22], which randomly selects a DQN head to greedily execute actions with the biggest Q value to achieve deep exploration, we also use some fixed low-level policy to execute a goal. Besides, ensemble can be leveraged to decrease approximation error variance in target Q value and makes the training of DQN more stable [5]. We use this trick in the TD3 algorithm with the ensemble. We update low-level critics with the target Q value below:

$$y = r^l + \gamma \frac{1}{K} \sum_{k=1}^{K} [\min_{i=1,2} Q_{\vartheta_i}^k(s', g', \pi_{\varphi'}(s', g') + clip(\mathcal{N}, -d, d))], \tag{12}$$

where ϑ', φ' denote the parameters of the target Critic network and the target Actor network respectively. With this, some poorly trained low-level policy cannot change the target Q value too much away from normal. Hence, the updating of low-level policies will be more stable. We train low-level policies respectively in each update step with mini-batches randomly sampled from the low-level experience replay buffer.

C. *Gates Module*

To estimate the success potential of each low-level policy at a specific state-goal pair, we design the Gates module to score each low policy. Different from the general Mixture-of-Experts (MoE) method [30], [31] which use only one gating network (learner) to decide which experts to use, we use K gates (scoring machines) to make up the Gates module:

$$Gates(s, g) = \{G_{\psi_k}(s, g)\}_{k=1}^{K} \tag{13}$$

Intuitively, a gate means a machine to score its related low-level policy. In our method, we use the negative Euclidean distance from the end state of a goal trajectory segment $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, s_{t+1:t+c})$ to the ideal goal state to score the current low-level policy:

$$score_k = ||g_t - f(s_{t+c} - s_t)||_2, \tag{14}$$

where k means the scoring machine of the k-th low-level policy. Therefore, we could build a database $\mathcal{B}_k^G = \{(s_i, g_i, score_k^i)\}_{i=1}^{N}$ for k-th low-level policy. Only one low-level policy could be picked to execute goal g_t at state s_t, so we cannot get the score of all K low-level policies at a specific state-goal pair. Therefore, using a single network to score all low-level policies is insufficient. Consequently, we use K gates to make up the Gates module and generate the scheduling vector $\boldsymbol{m_t}$, a K-dimension vector with which we select the highest scoring low-level policy. We update Gates according to:

$$\psi_k \leftarrow \arg\min_{\psi_k} \frac{1}{N'} \sum (score_k - G_{\psi_k}(s, g))^2, \tag{15}$$

where $k = 1, 2, ..., K$.

V. Experiments

In this section, we compare the performance of our method and state-of-the-art goal-conditioned HRL methods [9], [10]. The results show the better performance of our method.

A. Environmental Setup

We conduct our experiment on challenging long-horizon continuous control tasks [6] based on MuJoCo simulator. We choose the quadrupedal ant robot to complete the following tasks:

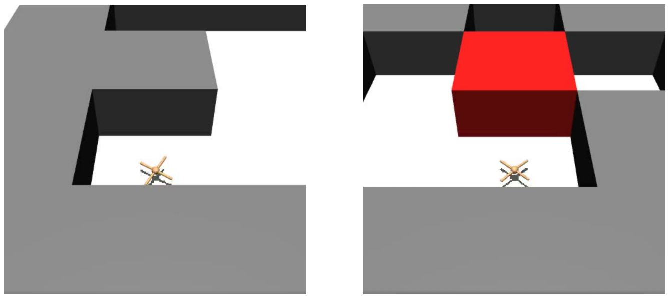

(a) AntMaze Env. (b) AntPush Env

Fig. 2. Experiment environments

Ant Maze: As shown in Fig. 2(a), the maze is U-shaped and closed. The ant robot needs to bypass the unmovable block and find a way to the top left corner from the bottom left corner. The high-level action is defined as position coordinates of the agent. The environment reward is set as the negative Euclidean distance from the current position (x, y) to the target position.

Ant Push: As shown in Fig. 2(b), there is a movable block between the initial position and the target position in this map. The ant robot can only push it but not pull it. The block will fill the target room if the ant walks directly to the target position. So the ant must learn to push the block to the side room and then walk to the target.

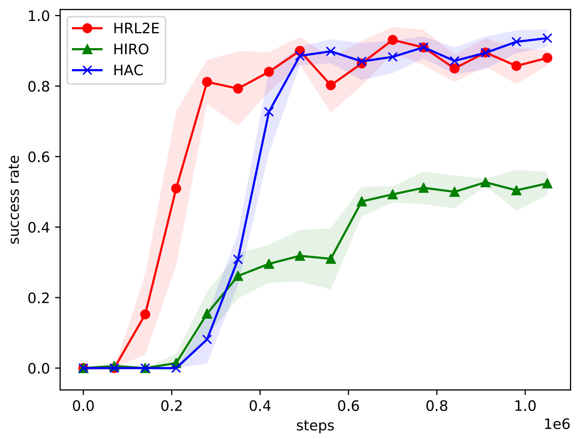
Each experiment is conducted five times over different random seeds of environment and networks. The time horizon is set 1000 for Ant Push and 1500 for Ant Maze. We set $K = 3$ for ensemble, $c = 10$ as the time budget for each high-level action, discount $\gamma = 0.99$, batch size 100. All networks are updated with Adam optimizer with a learning rate of $1e-4$ for actors, $1e-3$ for critics, and $1e-3$ for Gates. The soft update factor is 0.005. The low level updates every step, and the high level updates every 10 steps. The policy noise standard deviation for target policy smoothing regularization in TD3 is 1, so is the exploration noise of actions. Finally, we evaluate methods every $1e4$ steps, and in each evaluation, we evaluate it twenty times with 5 random seeds. If not specified, the above parameters shared by high and low levels are set the same. In different methods, we keep the shared parameter settings consistent.

We use these following methods as baselines:

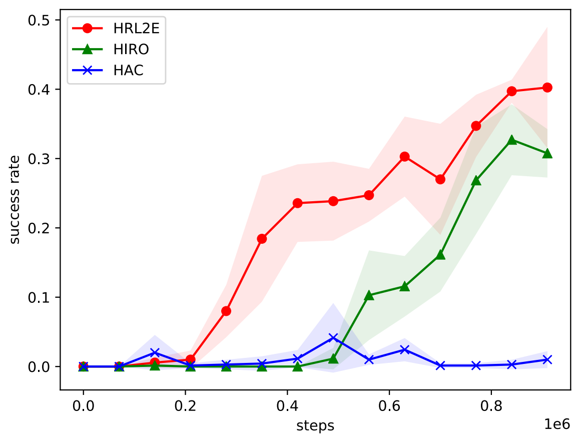
- HIRO [9], a state-of-the-art goal-conditioned HRL algorithm, using off-policy correction to alleviate non-stationary problem, training high-level policy with goals whose trajectories are most likely to be generated by current low-level policy.
- HAC [10], a state-of-the-art goal-conditioned HRL algorithm, using hindsight action transition to solve non-stationary problem, training high-level policy with goals mapped from the actually arrived end state within the timestep budget.

B. Results

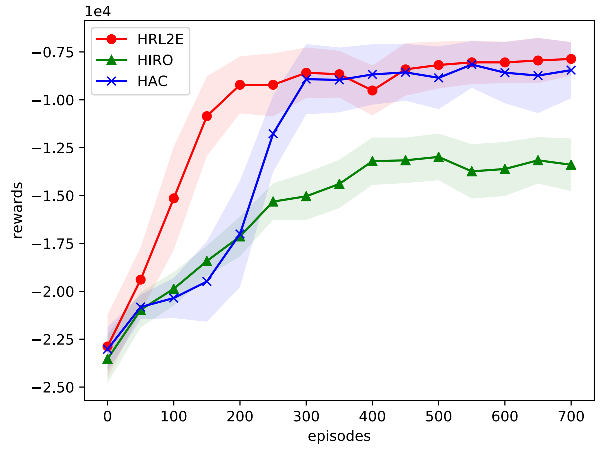
We show the comparison with baselines in Fig. 3. We get the curves in Fig. 3 by averaging 5 experiment curves. Fig. 3(a) and Fig. 3(b) present the curves of success rate in AntMaze and AntPush respectively. The rewards in each episode during training are presented in Fig. 3(c) for AntMaze and Fig. 3(d) for AntPush. As shown, our method HRL2E is the first one to start converging in both environments. In AntMaze, HRL2E first achieves a positive success rate at about $100k$ step, about 50% faster than HIRO and HAC. The reward curves of AntMaze also reflect that our method first find the trajectory with the maximum reward sum. In AntPush, HRL2E also costs about half as many steps as HIRO does to first achieve a positive success rate. Moreover, our method gets the peak success rate in this environment. However, with the same parameter settings, HAC can hardly accomplish this task in evaluation within $1M$ steps. We infer that the change of environment could influence HAC more than HIRO, so the performance of HAC could be unstable in



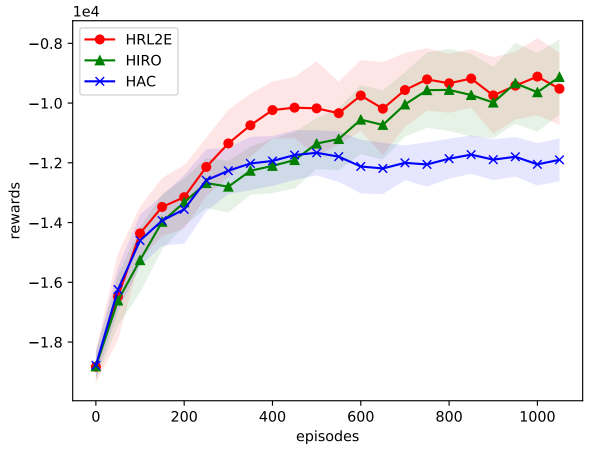
(a) Success rate in AntMaze



(b) Success rate in AntPush



(c) Rewards in AntMaze



(d) Rewards in AntPush

Fig. 3. Experiment results in AntMaze, AntPush

dynamic environments. The rewards curve in AntPush shows that our method can get the most rewards during training. In general, our method HRL2E performs more stably in both environments and can indeed accelerate convergence.

VI. Conclusion and Future Work

In this work, we introduce a goal-conditioned HRL method HRL2E, Hierarchical Reinforcement Learning with Low-level Ensemble, which leverages ensemble structure and Gates module to accelerate the non-stationary stage of HRL. Concretely, we train several low-level policies and the same amount of score machines to score each low-level policy, and we determine whether to pick it according to the score. Our experiments show that HRL2E indeed accelerates the convergence and outperforms state-of-the-art goal-conditioned HRL algorithms HIRO and HAC.

We use the Gates module to select the most proper low-level policy, while there are many other ways to schedule policies, such as upper-confidence bounds (UCB). Besides, we only consider leveraging ensemble in low level to strengthen its capacity. However, in many tasks, the exploration capacity of the high level may matters more. Therefore, leveraging ensemble in the high level to guide exploration is worth trying.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 3389–3396.

[3] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, "Why does hierarchy (sometimes) work so well in reinforcement learning?" *arXiv preprint arXiv:1909.10618*, 2019.

[4] R. Wang, R. Yu, B. An, and Z. Rabinovich, "I2hrl: Interactive influence-based hierarchical reinforcement learning," in *Proceedings of the 29th International Joint Conference on Artificial Intelligence.*, 2020, pp. 3131–3138.

[5] O. Anschel, N. Baram, and N. Shimkin, "Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 176–185.

[6] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1329–1338.

[7] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3540–3549.

[8] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[9] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *arXiv preprint arXiv:1805.08296*, 2018.

[10] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *arXiv preprint arXiv:1712.00948*, 2019.

[11] J. Kim, Y. Seo, and J. Shin, "Landmark-guided subgoal generation in hierarchical reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[12] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-optimal representation learning for hierarchical reinforcement learning," *arXiv preprint arXiv:1810.01257*, 2018.

[13] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, "Generating adjacency-constrained subgoals in hierarchical reinforcement learning," *arXiv preprint arXiv:2006.11485*, 2020.

[14] S. Li, R. Wang, M. Tang, and C. Zhang, "Hierarchical reinforcement learning with advantage-based auxiliary rewards," *arXiv preprint arXiv:1910.04450*, 2019.

[15] S. Sohn, J. Oh, and H. Lee, "Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies," *arXiv preprint arXiv:1807.07665*, 2018.

[16] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[17] M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup, "Learnings options end-to-end for continuous action tasks," *arXiv preprint arXiv:1712.00004*, 2017.

[18] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *International Conference on Learning Representations*, 2019.

[19] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," *arXiv preprint arXiv:1910.11956*, 2019.

[20] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.

[21] R. Y. Chen, S. Sidor, P. Abbeel, and J. Schulman, "Ucb exploration via q-ensembles," *arXiv preprint arXiv:1706.01502*, 2017.

[22] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," *Advances in Neural Information Processing Systems*, vol. 29, pp. 4026–4034, 2016.

[23] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, "Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6131–6141.

[24] S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek, "End-to-end hierarchical reinforcement learning with integrated subgoal discovery," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[25] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Exploration–exploitation tradeoff using variance estimates in multi-armed bandits," *Theoretical Computer Science*, vol. 410, no. 19, pp. 1876–1902, 2009.

[26] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[27] Z. Zheng, C. Yuan, X. Zhu, Z. Lin, Y. Cheng, C. Shi, and J. Ye, "Self-supervised mixture-of-experts by uncertainty estimation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 5933–5940.

[28] D. A. Nix and A. S. Weigend, "Estimating the mean and variance of the target probability distribution," in *Proceedings of 1994 IEEE International Conference on Neural Networks*, vol. 1. IEEE, 1994, pp. 55–60.

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[30] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[31] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural Computation*, vol. 6, no. 2, pp. 181–214, 1994.

IJCNN 2022

IEEE WCCI 2022
IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE
PADUA, ITALY • 18 - 23 July, 2022

HRL2E: Hierarchical Reinforcement Learning with Low-level Ensemble

You Qin (presenter)
with Zhi Wang and Chunlin Chen
Email: youqin@smail.nju.edu.cn

Department of Control and Systems Engineering,
School of Management and Engineering, Nanjing University