Competitive and Cooperative Heterogeneous Deep Reinforcement Learning

Han Zheng
University of Technology Sydney
Sydney, Australia
Han.Zheng-1@student.uts.edu.au

Jing Jiang
University of Technology Sydney
Sydney, Australia
Jing.Jiang@uts.edu.au

Pengfei Wei
National University of Singapore
Singapore
dcsweip@nus.edu.sg

Guodong Long
University of Technology Sydney
Sydney, Australia
Guodong.Long@uts.edu.au

Chengqi Zhang
University of Technology Sydney
Sydney, Australia
chengqi.Zhang@uts.edu.au

ABSTRACT

Numerous deep reinforcement learning methods have been proposed, including deterministic, stochastic, and evolutionary-based hybrid methods. However, among these various methodologies, there is no clear winner that consistently outperforms the others in every task in terms of effective exploration, sample efficiency, and stability. In this work, we present a competitive and cooperative heterogeneous deep reinforcement learning framework called C2HRL. C2HRL aims to learn a superior agent that exceeds the capabilities of the individual agent in an agent pool through two agent management mechanisms: one competitive, the other cooperative. The competitive mechanism forces agents to compete for computing resources and to explore and exploit diverse regions of the solution space. To support this strategy, resources are distributed to the most suitable agent for that specific task and random seed setting, which results in better sample efficiency and stability. The other mechanic, cooperation, asks heterogeneous agents to share their exploration experiences so that all agents can learn from a diverse set of policies. The experiences are stored in a two-level replay buffer and the result is an overall more effective exploration strategy. We evaluated C2HRL on a range of continuous control tasks from the benchmark Mujoco. The experimental results demonstrate that C2HRL has better sample efficiency and greater stability than three state-of-the-art DRL baselines.

KEYWORDS

deep reinforcement learning, heterogeneous agents, competition and cooperation

ACM Reference Format:

Han Zheng, Jing Jiang, Pengfei Wei, Guodong Long, and Chengqi Zhang. 2020. Competitive and Cooperative Heterogeneous Deep Reinforcement Learning. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Deep reinforcement learning (DRL) is a learning strategy combining deep learning [39] and reinforcement learning [36]. It has achieved promising results in numerous challenging real-world problems, e.g., AI games [23], constraint satisfaction problems [33] and robotic control [1]. However, existing DRL algorithms usually require a huge training cost (including large amount of training data, the powerful computing resources, and long training phase) to achieve satisfactory performance. This is because they suffer from two major limitations: (1) the lack of effective exploration and (2) high sample complexity.

Exploration is a key component of an agent's ability to learn a good policy and avoid converging to the local optima prematurely. Various exploration strategies have been proposed, e.g., noise-based exploration [8], information maximizing exploration [14], count-based exploration [24, 37], curiosity-driven exploration [25] and intrinsic motivation exploration [3]. Each of them has achieved promising exploration efficiency in some specific tasks. However, it is unclear which strategy should be given the priority since none of them consistently outperforms the others in various tasks. This is probably because these exploration strategies are effective for some tasks but not for other tasks. A general exploration strategy that is universally appropriate across different tasks and learning algorithms remains a big outstanding challenge.

Sample complexity and stability are other significant challenges for the DRL agent. Policy-based DRL methods, e.g., TRPO [29], PPO [30], and A3C [22], are spectacularly sample-expensive due to on-policy learning. They require new samples to be collected in each gradient step. Off-policy learning methods [36] improve the sample efficiency by reusing the past experiences. Nevertheless, the combination of off-policy learning with high-dimensional and non-linear function approximation by deep neural networks presents a significant challenge for convergence and stability [4]. What

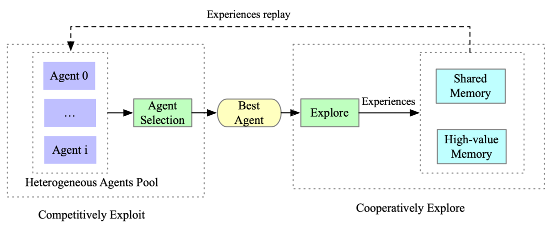


Figure 1: The high-level structure of C2HRL for one iteration

is more problematic is that some researchers have demonstrated that deceptive gradient information and random seeds drastically affect learning performance [12, 26]. Given a learning task, how to achieve high sample efficiency while maintain stability across different random seeds is not well studied.

In this paper, we explore how to design an efficient and stable algorithm for continuous state and action spaces across different tasks and random seed settings. Our aim is to propose a learning diagram that consistently enables effective exploration and sample efficiency in various learning tasks. To this end, we introduce a Competitive and Cooperative Heterogeneous Deep Reinforcement Learning (C2HRL) framework. C2HRL is a scalable framework that leverages the advantages of different state-of-the-art gradient-based DRL agents including deterministic-policy agent[9], stochastic-policy[11] agent as well as gradient-free agent (learning based on Evolutionary-Algorithms(EAs)[7]) to handle diverse DRL tasks. Specifically, we propose a cooperative exploration mechanism that forces different agents to explore the action and state space in a collaborative way. This is done by sharing the exploration experiences among different agents. Moreover, to guarantee sample efficiency, we propose a competitive mechanism to dynamically select the most promising agent in each training iteration, and distribute the most computing resources to it. To implement this strategy, we propose a new metric, called growth capacity, that measures the potential of the agent on the growth of the cumulative returns. Using this growth capacity metric, the resource manager continually evaluates different agents and selects the best one in each training iteration. Note that this competition mechanism also leads to more stable performance across various random seeds.

Figure 1 illustrates the high-level structure of the C2HRL framework. The agent pool contains a selection of heterogeneous agents, e.g., TD3 [9], SAC [11], and EAs [21]. During the cooperative exploration phase, all the agents store their exploration experience to a global shared memory buffer using their own exploration policies. At the same time, we store episode with high value into a high value memory to make agents learn more effectively from diverse experiences. Regarding the competitive phase, an agent manager calculates the growth capacity of all the agents and selects the current best agent to exploit for the next iteration. This above procedure iterates until termination.

In experiments to evaluate the efficiency of C2HRL, we find our method outperformed three state-of-the-art baselines – SAC [11], TD3 [9], and CERL [15] – in a range of continuous control benchmark tasks.

In summary, the contributions of this research are as follows:

(1) We propose a scalable framework: C2HRL, that takes advantages of diverse agents, including off-policy RL agents and EA agents, to achieve a better performance.
(2) We propose a combined cooperative and competitive mechanism among heterogeneous agents to improve the model's exploration effectiveness and stability.
(3) We present empirical results that show our model outperforms three baselines in a range of continuous control benchmarks.

2 BACKGROUND

This section begins with an introduction to the basic concept of RL and evolutionary algorithms. We then review two state-of-the-art RL methods: twin delayed deep deterministic policy gradients (TD3) [9], soft actor-critic (SAC) [11].

2.1 Reinforcement Learning

In a standard RL problem, the interaction between an agent and an environment e is modeled as a Markov decision process. At each time step t, the agent observes a state s_t and chooses an action $a_t \in \mathcal{A}$ using a policy $\pi(a_t|s_t)$ that maps states to a distribution over possible actions. In this paper, we are concerned with high-dimensional, continuous state and action spaces. After performing an action a_t in each time step, the agent collects a reward $r(s_t, a_t) \in \mathcal{R}$. The objective in RL is to learn a policy that maximizes the expected sum of discounted rewards starting from the initial state. The objective is shown below:

$$J(\pi) = \mathrm{E}_{(s_t,a_t)\sim e,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \tag{1}$$

where s_t represents the state that is sampled from the environment e at time step t using an unknown system dynamic model $p(s_{t+1}|s_t, a_t)$ and an initial state distribution $p(s_0)$,and a_t represents the action that is sampled from the policy $\pi(a_t|s_t)$ at time step t. $\gamma \in (0, 1]$ is the discount factor used to compute the sum of all rewards ever obtained by the agent, but discounted by how far off in the future they are obtained.

2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a class of black-box search algorithms that apply heuristic search procedures inspired by natural evolution. EAs typically consist of three main operators: new solution generation, solution alteration, and selection [7, 34]. In general, these operations are applied to a population of candidate solutions, which produce next-generation solutions while keeping the promising ones from the previous generation. The selection operation is probabilistic, where solutions with higher fitness values have a higher probability of being selected. Assuming the higher fitness values are representative of good solution quality, the overall quality of solutions should improve with each passing generation. In this work, each individual in EA defines a deep neural network.

"Mutations" are random perturbations to the parameters of these neural networks. The evolutionary framework used here is closely related to evolving neural networks and is often referred to neuroevolution [6, 21, 35].

2.3 Twin Delayed Deep Deterministic Policy Gradients(TD3)

TD3 [9] is a method based on an actor-critic architecture that alleviates the issue of overestimating values and sub-optimal policies caused by function approximation errors. TD3 is an extension to DDPG [19] that learns two Q-functions, i.e., Q1 and Q2, by minimizing the mean square Bellman error. It improves upon DDPG in the following three respects.

Target policy smoothing regularization. TD3 adds a clipped noise to each dimension of the target action that is based on a target policy $\mu_{\theta_{targ}}$. Then, the noisy target action is clipped to stay in the valid action range:

$$a'(s') = \text{clip}(\mu_{\theta_{targ}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}), \epsilon \sim \mathcal{N}(0, \sigma). \tag{2}$$

This regularization method addresses the concerns that deterministic policies may overfit to narrow peaks in the value estimation. This can be avoided by smoothing out the Q-value over similar actions.

Clipped double-Q learning. TD3 uses the smaller Q-value for the target:

$$y(r, s', d) = r + \gamma(1 - d)\min_{i=1,2} Q_{\phi_i,targ}(s', a'(s')). \tag{3}$$

By doing so, TD3 avoids the overestimating the Q-value function.

Delayed policy updates. An overestimated or inaccurate value makes the value estimation diverge, resulting in poor policy learning. Hence, TD3 only updates the policy when the error in value is sufficiently small. The update is done by maximizing Q_{ϕ_1}:

$$\max_{\theta} E_{s\sim D}[Q_{\phi_1}(s, \mu_\theta(s))]. \tag{4}$$

2.4 Soft Actor-critic(SAC)

SAC [11] incorporates an entropy measure of the policy into the reward to encourage exploration. The intuition is to learn a policy that acts as randomly as possible while still being able to succeed in the task. It is an off-policy actor-critic model that follows the maximum entropy RL framework. The policy is trained with the objective of maximizing the expected return and the entropy at the same time:

$$J(\pi) = \sum_{t=0}^{T} \mathrm{E}_{(s_t,a_t)} \sim \rho_\pi[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{5}$$

where $\mathcal{H}(\cdot)$ is the entropy measure and α controls how important the entropy term is, known as temperature parameter. Entropy maximization leads to policies that can: (1) explore more space and (2) capture multiple modes of near-optimal strategies. For example, if there exist multiple options that seem to be equally good, the policy should assign each with an equal probability of being chosen.

3 RELATED WORK

Our method incorporates two key elements: cooperative exploration and competitive exploitation. Cooperative exploration is mainly implemented through an experience replay mechanism [20], which is widely-used in off-policy reinforcement learning. DQN [23] randomly and uniformly samples the experience from a replay memory. [28] subsequently expanded DQN to develop prioritized experience replay (PER), which uses a temporal difference error to prioritize the experiences. Ape-x [13] extends PER to the distributed setting. [1] introduces a technique called Hindsight Experience Replay (HER), which allows sample-efficient learning from sparer and binary rewards. CERL [15] and ERL [16] employ a shared memory to collect data generated by a diverse set of actors. In[40], the authors introduce an episodic control experience replay method to latch on good trajectories rapidly. However, these methods only study the shared experiences of one type of agent, i.e., the same behaviour policy architecture. In other words, all experiences are generated by the same type of policy actor. Our method explores agents with different policy architectures and different learning algorithms. Through this, we can achieve a more effective exploration.

C2HRL's competitive mechanism can be discussed in terms of algorithm selection [10, 27, 32], which has been widely examined in the literature. In [17], Lagoudakis and Littman described an algorithm selection method that formulates the problem as a Markov decision process and draws on ideas from RL to solve that problem. Cauwet et al. [5] provide a noisy optimization method for a portfolio of solvers, achieving a similar result to the best solver. In [31], the authors apply a goal-switching method for policy selection. Laroche and Feraud [18] formalized the problem of online algorithm selection in the context of RL, presenting a selection algorithm: epochal stochastic bandit algorithm selection. The common thread in these works is that they solely focus on RL-based methods, whereas our C2HRL framework combines gradient-based RL agents with a gradient-free EA agent.

4 MOTIVATING EXAMPLE

4.1 Effective Exploratioin

In some random seed settings, some RL agents, such as TD3, may fail to learn because they cannot explore the solution space effectively. Figure 2 shows how a TD3 agent prematurely converges to a local minimum because of its ineffective initial exploration. In this case, we set the Mujoco' action-space seed to make TD3's initial sample actions are deterministic too. However, if the TD3 agent were to cooperate by learning from the shared exploration experiences of all agents, that TD3 agent may have a chance to succeed in learning and so perform well.

4.2 Sample Efficiency

The sample efficiency of algorithms varies greatly among tasks. Here, the sample efficiency is a concept to explain that how good an agent can utilize the exploration samples. Higher sample efficiency means a higher final average return. Table 1 shows the final performance of three algorithms on three different tasks. The gradient-free method, i.e., EA, learns more efficiently than other

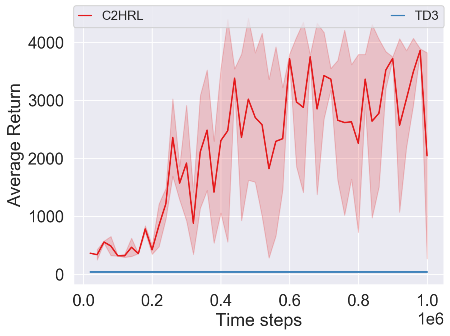


Figure 2: Hopper, Td3 fails to learn in a random seed setting, but does learn effectively within the C2HRL framework.

agents on the Swimmer task, while TD3 is the most efficient on Walker2d, and SAC is best on Humanoid. This demonstrates how a competition mechanism would be useful for distributing available resources to the most suitable learning method based on the current context, whether a task, random seed setting, etc.

Name	Swimmer	Humanoid	Walker2d
TD3	69	457	**5701**
SAC	45	**5686**	5087
EA	**350**	1100	1200

Table 1: The efficiency of different algorithms on one seed of three different tasks. The score is the maximum average return over 5 episodes trials for 1 million training steps.

5 COMPETITIVE AND COOPERATIVE HETEROGENEOUS DEEP REINFORCEMENT LEARNING(C2HRL)

The principal idea behind this work is to combine the strengths of multiple heterogeneous agents, where different agents may have different exploration and learning strategies. For instance, TD3 exploits a deterministic policy learning strategy, while SAC employs a stochastic one. Given diverse continuous tasks in a dynamic environment, a specific agent is unlikely to always be optimal for all the tasks. Even for a single task, it is highly preferable to dynamically adapt the agent to the environment to suit the learning task. To accomplish our goal, we propose C2HRL – a competitive and cooperative heterogeneous reinforcement learning framework. C2HRL is built on two fundamental mechanisms, namely, competitive exploitation and cooperative exploration. The competitive exploitation mechanism leverages the fact that different agents posses different learning potentials – some agents learn quickly and prematurely converge to a local optima, while other agents learn slowly in the beginning but yield much better performance in the end. To incorporate the benefits of different learning potentials, C2HRL dynamically and adaptively selects the best agent among multiple alternatives in each training iteration. The cooperative exploration mechanism ensures that the different agents benefit from all the different exploration policies. As different exploration policies may cover different crucial parts of the search space, a collaborative approach promotes a more efficient and complete exploration. C2HRL is presented in detail in Algorithm 1.

5.1 Competitive Exploitation

In this section, we explain the competition mechanism in C2HRL. The first step is to create an agent pool containing n agents. This agent pool not only includes heterogeneous agents that use different exploration and learning strategies, e.g., TD3, SAC, and EA, but it may also contain homogeneous agents where multiple agents use the same learning strategy but with different hyperparameters, e.g., TD3 with different discount factors. Note that the latest work, CERL, only works for the latter. Within a fixed number of timesteps T, i.e., one iteration, where one timestep represents one interaction with the environment, the best agent is selected. Note that one iteration contains p roll-outs, i.e., p episodes of interaction with the environment. To identify the best agent in each iteration, the performance of every agent needs to be evaluated for its exploration and exploitation efficiencies. A widely-used metric that provides a good trade-off between exploration and exploitation is the upper confidence bound (UCB) [2]. The classic UCB, used in [15], is formally defined as:

$$U_i^j = \hat{v}_i^j + c * \sqrt{\frac{\log(\sum_{i=1}^{b} y_i^j)}{y_i^j}} \qquad (6)$$
$$v_i^j \leftarrow \alpha * r_i^j + (1-\alpha) * v_i^{j-1}$$

where U_i^j is the UCB score of the i-th agent in the j-th iteration, b is the number of agents, y_i^j is the number of cumulative roll-outs the i-th agent has run in the j iterations, $\hat{v}_i^j$ is the discounted sum of the cumulative returns received from y_i^j roll-outs and it is normalized to be $\in (0,1)$, r_i^j is the return of the j-th iteration, and α and c are the balancing parameters.

As seen from Eq. (6), the UCB only uses the cumulative return of the existing iterations to evaluate agent performance; it ignores potential performance variations in the following iterations. For instance, some agents start with a very promising cumulative return, but quickly converge. In this case, it is desirable to select only these agents in the first several iterations, and make adjustments to other agents that may have a lower cumulative return but higher return growth in the following iterations. To take the return growth into account in the evaluation, we defined a value metric, called growth capacity, that measures the potential of an agent to increase the cumulative returns. Formally, this metric is defined as the temporal difference between the returns in adjacent iterations:

$$g_i^j = \mu * (r_i^j - r_i^{j-1}) \qquad (7)$$

where μ is a normalization factor to avoid extremely large values. Note that Eq. (7) calculates growth capacity uniformly in all iterations. However, for more intensive exploration, it is usually much more desirable to increase the diversity of the selected best agent in the early stages, while maintaining the stability of the selected best agent in the later stages to guarantee convergence. This motivated us to refine the growth capacity measure as follows:

$$\hat{g}_i^j = t_i^j / T_m * g_i^j \tag{8}$$

where T_m is the number of complete time steps and t_i^j is the number of steps that the i-th agent has run in the j iterations. The more times the i-th agent is selected as the best agent in the j iterations, the larger t_i^j is. In the early stages, all the agents have a small t, and thus C2HRL encourages the variety in the best agent selected. During the training process, those agents that are selected more times in the previous steps accumulate larger t, and thus C2HRL tends to preserve these agents. With the growth capacity defined in Eq. (8), the UCB score is refined as follows:

$$\begin{aligned} u_i^j &= \hat{p}_i^j + c * \sqrt{\frac{\log(\sum_{i=1}^{b} y_i^j)}{y_i^j}}, \\ p_i^j &\leftarrow \alpha * \hat{g}_i^j + (1-\alpha) * p_i^{j-1}. \end{aligned} \tag{9}$$

We then use Eq. (9) as the evaluation metric to measure the performance of an agent in one iteration. A greedy strategy is then applied to select the agent with the largest UCB score. The selected agent is allocated the computing resources that it needs, e.g., TD3 or SAC only needs one actor, and the other agents will free the computing resources.

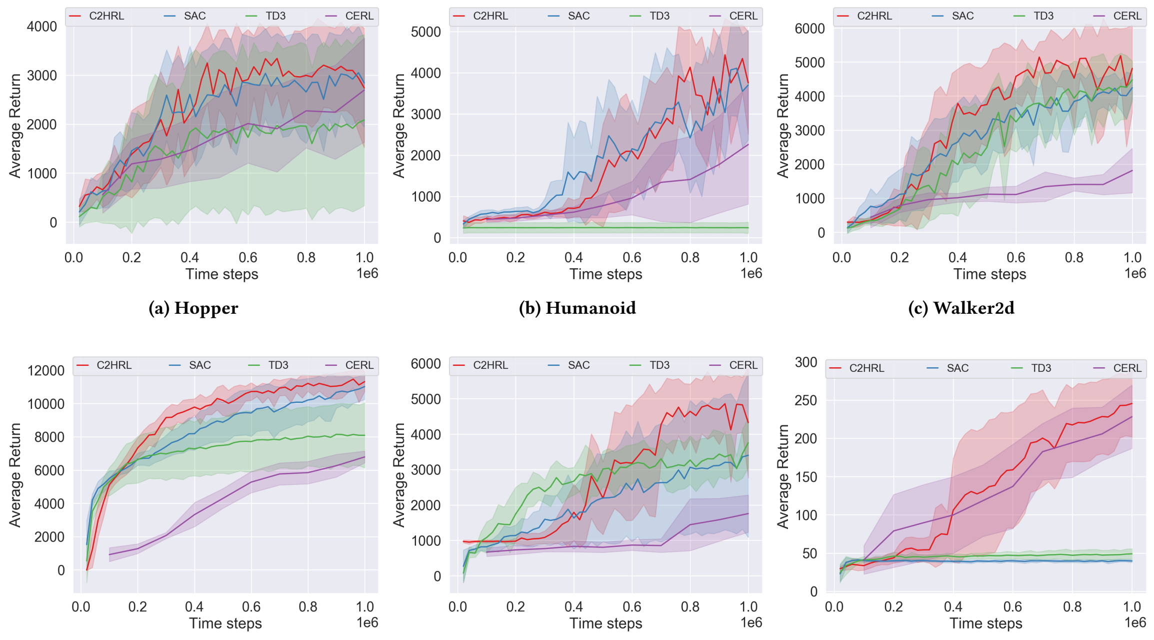
5.2 Cooperative Exploration

Although C2HRL encourages different agents to compete for the same resources, it also incorporates a cooperation mechanism to make exploration more effective through a two-level shared memory buffer that stores the experiences of different agents in the learning procedure. It is worth noting that these experiences come from different exploration policies. Any of them alone may be ill-suited to solving the current task, but they may also provide crucial knowledge on specific parts of the search space. Integrating all these experiences together helps to generate a diverse and complete view of the search space, which may be the key to learning a task well. With this approach, C2HRL not only learns more effectively, it also helps to expose elite agents more quickly, which leads to a higher sample efficiency.

With AEDDPG, Sunehag et al. [40] introduced the concept of episodic memory to off-policy DRL. The strategy is to store an episode in a new high-value memory buffer if the episode has a higher return than the historical maximum return. However, AEDDPG only works with one type of DRL agent and would likely fail with a diverse group of agents and learning strategies. Additionally, in a setting with multiple heterogeneous agents, the population for a gradient-free EA could grow very quickly, generating a very high episodic return reward in the early stages. But this would lead to a high-value memory buffer filled with the experiences of EA agents and very few experiences from the others. Hence, rather than using the historical maximum return, we have opted to exploit

Algorithm 1: C2HRL Algorithm

```
1  ‘ Initialize the agents pool $\mathcal{P}$: $a_0, a_1, ...a_n$, RL sample
     probability $p_0, ...p_n$ from high-value memory, initial
     explore timesteps $t_0, ..., t_n$, maximum steps $T_m$
2  Initialize iteration steps to $T$,normalization factor $\mu$,a
     random number generator $r() \in [0, 1]$
3  Initialize agents' status $\mathcal{S}$, shared memory $M$, high value
     memory $HM$, high-value threshold $h_t$
4  Initialize agents' current max fitness $\mathcal{F}$, start-competition
     generations $\mathcal{G}$, generations $g$, EA threshold value $F_t$
5  while not finished do
6      for $a_i \in \mathcal{P}, i \in [0, n]$ do
7          if $a_i$ chosen or $g < \mathcal{G}$ then
8              Explore one iteration $T$ steps and get
                 experiences $\mathcal{E}$, cumulative return $f$
9              Update explore steps $t_i + = T$
10             Store_Experiences($\mathcal{E}, f, M, HM$)
11             Learn($a_i, t_i, M, HM$)
12         Update_Status($\mathcal{S}, i, f, t$)
13     end
14     Choose the agent with max UCB score for next iteration
15     $g + = 1$
16 end
17 Store_Experiences ($\mathcal{E}, f, M, HM$):
18     Store $\mathcal{E}$ to $M$
19     if RL agent then
20         if $f >= min(\mathcal{F})$ then
21             Store $\mathcal{E}$ to $HM$
22     else
23         get maximum value of last generation: $f'$
24         if $f >= min(\mathcal{F})$ and $(f - f') > F_t$ then
25             Store $\mathcal{E}$ to $HM$
26     return
27 Update_Status ($\mathcal{S}$,$i$,$f$,$t$):
28     $f = (f - \mathcal{F}[i]) * \mu$
29     $\hat{g}_i = f * (t/T_m)$
30     if $\hat{g}_i > \mathcal{S}[i]["growth"]$ then
31         $\mathcal{S}[i]["growth"] \leftarrow \alpha * \hat{g}_i + (1 - \alpha) * \mathcal{S}[i]["growth"]$
32         $\mathcal{F}[i] = f$
33     return
34 Learn ($a_i, p_i, M, HM$):
35     if $a_i$ is RL agent then
36         for each learning step do
37             if $r() < p_i$ and HM size $> h_t$ then
38                 Sample mini-batch experiences $b$ from $HM$
39             else
40                 Sample mini-batch experiences $b$ from $M$
41             Using $b$ to update agent $a_i$ based on its
                 gradient-based learning method
42         end
43     else
44         Update agent $a_i$ based on its gradient-free learning
             method
45     return
```



(a) Hopper (b) Humanoid (c) Walker2d

(d) HalfCheetah (e) Ant (f) Swimmer

Figure 3: Training curves on Mujoco continuous control tasks.

Environment	C2HRL	CERL	SAC	TD3
Humanoid-v2	**5300**±**98**	3417±1514	5206±131	249±142
Ant-v2	**5680**±**283**	1767±516	3561±2319	4608±1199
Walker2d-v2	**5818**±**212**	1928±580	4566±513	4778±724
Hopper-v2	**3741**±**125**	2800±831	3599±139	2126±1923
HalfCheetah-v2	**11969**±**207**	6786±355	11346±445	9980±957
Swimmer-v2	**245**±43	227±41	43±**3**	55±9

Table 2: Max average return over 5 trials of 1 million time steps. Bold indicates the maximum average value for each task. ± indicates the standard deviation.

the minimum return of all the agents in each episode as the value to decide whether to store an episode into high-value memory or not.

On the other hand, the EA population maintains the elite individuals, which may have the same parameters compared with the last generation's elites. To avoid storing similar experiences into high-value memory, a threshold value F_t is used to determine whether to save the current experiences to high-value memory. Particularly, if the current fitness is greater than the maximum of the last generation more than F_t, current experiences will be stored in high-value memory. Moreover, the sample probability from high-value memory is different from [40]. In our method, different agents sample from high-value memory according to a different probability instead of using the same one used in [40]. We found that the deterministic RL agent learns better from high-value memory than the stochastic agent. Therefore, the sample probability of TD3 is higher than SAC in our setting. To make exploration more effective, there is no competition during the initial $\mathcal{G}$ generations, and RL agents learn from high-value memory only when high-value stores enough experiences ($> h_t$).

6 EXPERIMENTS

The goal of our experimental evaluation was to verify the sample efficiency and stability of C2HRL. To that end, we compared C2HRL with several state-of-the-art DRL methods. We also conducted an ablation study to investigate the influence of each component in C2HRL. All the evaluations were conducted on continuous control benchmark: Mujoco [38].

6.1 Comparative Evaluation

We first evaluated C2HRL's performance on six continuous control tasks from Mujoco in comparison to three state-of-the-art baselines – SAC [11], TD3 [9], and CERL [15]. For SAC, we use the code from OpenAISpinningUp [1]. For TD3 and CERL, we used the author-provided implementation, and we used the default hyperparameters as outlined in the corresponding papers. Note that we also used SAC and TD3 as candidate agents in our agent pool for C2HRL in addition to the EA method and associated hyperparameters in [16]. For C2HRL's specific hyperparamters, the TD3 sample probability p_t from high-value memory is 0.4, while SAC's p_s is 0.3. The size of high-value memory size is 20000. The initial fairly competition generations $\mathcal{G}$ is 2. The threshold size h_t of high-value memory is 10000. EA threshold value F_t is 10. Iteration time steps T is 10000. maximum time steps T_m is $1e6$.

We ran the training process for all the methods over 1 million time steps on each task with five different seeds. One time step represents one interaction between the agent and the environment. Learning performance is reported as the average return of five independent trials of each seed, taking the mean of the five seeds as the final score. For C2HRL, we set 10000 time steps as one iteration for the best agent selection. We report the scores of all the methods compared against the number of time steps.

Figure 3 shows the comparative results for all methods on all six Mujoco learning tasks. From the results, we first observe that there was no clear winner among the existing state-of-the-art baselines SAC, TD3, and CERL. None consistently outperforms the others on the six learning tasks. Specifically, SAC outperforms on the first, second and fourth tasks, TD3 wins on the third and fifth tasks, while CERL only yields the best performance on the last task but achieves significant improvements compared with the other two. This verifies the challenging issue in the current DRL study, that is, it lacks of a general exploration strategy universally appropriate across different tasks. Fortunately, Figure 3 further demonstrates that C2HRL consistently performs better results than the best baseline methods on all six tasks, which highly alleviates the above issue. At the beginning of the learning phrase, C2HRL concentrates on intensive exploration by encouraging competition among different candidate agents. This leads to a relatively slow learning speed, but results in a better final performance due to a more diverse and effective exploration effort.

Table 2 provides the max average return as well as the corresponding standard deviation of the five independent trials across five random seeds. As can be seen from the results, C2HRL achieved the best average performance and smallest standard deviation in almost all six tasks, which again verifies that C2HRL provides a more stable learning performance among different random seed settings.

6.2 Competitive Resource Redistribution

In this section, we investigated how C2HRL distributes resources across different random seeds. In C2HRL, computing resources are dynamically distributed to different agent throughout the entire learning process. This distribution can differ dramatically in different random seed, even within the same learning task. This enables C2HRL to maintain a stable performance across different random seeds.

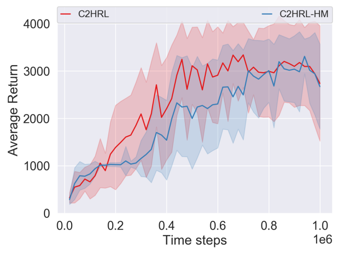
[1]https://github.com/openai/spinningup

For this experiment, we again included three different agents in C2HRL – EA, SAC, and TD3 – and analyzed the resource distribution rate across different random seeds (from 0 to 4) and across the six learning tasks. The results are given in Table 3. From the table, we can see clear diversity in the resource distribution. In the test with the different tasks but the same random seed, C2HRL distributed the most resources to different agents. For instance, in seed 0 of all tasks, SAC received the most computing resource for the Humanoid, Ant, and HalfCheetah tasks, followed by TD3, which was allocated the most for the Walker2d and Hopper tasks. EA only received the most resources for the Swimmer task. We also observed that C2HRL distributed the resources in a different manner when different random seeds were asked to perform the same task. Here, in the Hopper task, C2HRL distributed the most resources to TD3 in seeds 0, 1, and 4, but to SAC in seeds 2 and 3. This supports Henderson et al.'s [12] conclusion that random seeds indeed have a significant impact on agent performance. All the results demonstrate that C2HRL can adaptively distribute the resources among different agents in order to achieve the best learning performance.

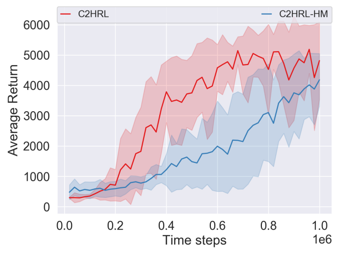
6.3 Ablation Studies

In this section, we conducted ablation studies to understand the contributions of each individual key component of C2HRL: cooperative exploration and competitive exploitation. To do this, we built two variants of C2HRL: C2HRL without the shared high-value memory (C2HRL-HM) and C2HRL without our growth capacity (C2HRL-GC). As mentioned, C2HRL achieves cooperative exploration by integrating the exploration experiences of all agents into two memory buffers, i.e., shared high-value memory and shared memory, so as to reuse these experiences in subsequent learning phases. Thus, C2HRL-HM is C2HRL without the shared high-value memory. C2HRL-GC is C2HRL without our growth capacity metric driving the competition mechanic but with conventional UCB instead, which only considers the immediate fitness. These variants plus the full C2HRL were tested on the Walker2d and Hopper tasks, and the comparative results are provided in Table 4; the performance metric used is the same as in Table 2.

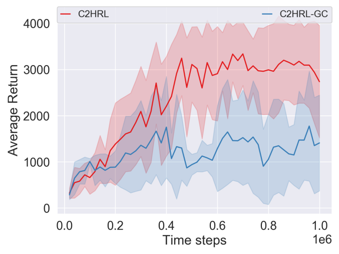
As shown in Figure 4, C2HRL achieved better results than both C2HRL-HM and C2HRL-GC in terms of both average performance and standard derivation. This indicates the superiority of combining the two key elements, i.e., the cooperative exploration and competitive exploitation. We further draw the learning curves for C2HRL-HM and C2HRL-GC, and compared them with that of C2HRL. Figure 4a and Figure 4b show the comparison of C2HRL-HM with C2HRL, and Figure 4c and Figure 4d show that of C2HRL-GC with C2HRL. It can be observed that C2HRL achieves slightly worse results than C2HRL-HM and C2HRL-GC at very first learning steps (before 0.2 million steps). This may be because C2HRL needs to distribute similar computing resources to different agents for better exploration in the beginning, and thus is less sample-efficient. Afterwards, C2HRL became better and better in comparison to C2HRL-HM and C2HRL-GC for the remainder of the learning phase. This verifies the effectiveness of the cooperative exploration and competitive exploitation across the entire dynamic learning process.



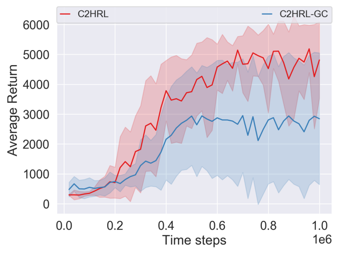
(a) Hopper-HM



(b) Walker2d-HM



(c) Hopper-GC



(d) Walker2d-GC

Figure 4: C2HRL-HM and C2HRL-GC

Agents	Humanoid					Ant					Walker2d					Hopper					HalfCheetah					Swimmer				
Seed	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
EA	0.07	0.09	0.08	0.09	0.07	0.05	0.05	0.05	0.06	0.06	0.12	0.03	0.06	0.05	0.05	0.03	0.02	0.12	0.07	0.04	0.01	0.01	0.01	0.01	0.01	**0.64**	**0.78**	**0.65**	**0.75**	**0.58**
SAC	**0.87**	**0.82**	**0.85**	**0.83**	**0.87**	**0.86**	**0.88**	0.07	**0.81**	**0.81**	0.07	0.04	0.05	0.05	0.09	0.02	0.02	**0.82**	**0.88**	0.05	**0.97**	**0.97**	0.01	0.01	**0.98**	0.18	0.11	0.17	0.12	0.21
TD3	0.06	0.09	0.08	0.08	0.06	0.09	0.07	**0.87**	0.13	0.13	**0.81**	**0.93**	**0.88**	**0.89**	**0.86**	**0.95**	**0.97**	0.05	0.04	**0.90**	0.01	0.01	**0.98**	**0.97**	0.01	0.18	0.11	0.18	0.13	0.21

Table 3: Resource distribution rate for C2HRL across tasks and random seeds.

Environment	Walker2d	Hopper
C2HRL	**5818±212**	**3741±125**
C2HRL-HM	4511±917	3530±140
C2HRL-GC	3270±2030	2490±1323

Table 4: The comparative results of C2HRL with C2HRL-HM and C2HRL-GC. The performance is the max average return over 5 trials of 1 million time steps. The best result for each task is highlighted in bold.

7 CONCLUSION

In this paper, we presented C2HRL, a scalable framework that allows gradient-based RL learners and gradient-free EA learners to jointly explore and exploit solutions for DRL problems. Experiments in a range of continuous control tasks demonstrate that C2HRL can outperform other baselines in both sample-efficiency and stability. In terms of the limitations of C2HRL, as C2HRL is trained based on multiple selected agents, its final performance depends on the construction of agent pool. Moreover, C2HRL introduces new hyperparameters, such as the sample probability from high-value memory. Future work will extend it to an adaptive setting method.

REFERENCES

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

[2] Peter Auer. 2002. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* 3, Nov (2002), 397–422.

[3] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*. 1471–1479.

[4] Shalabh Bhatnagar, Doina Precup, David Silver, Richard S Sutton, Hamid R Maei, and Csaba Szepesvári. 2009. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*. 1204–1212.

[5] Marie-Liesse Cauwet, Jialin Liu, Baptiste Rozière, and Olivier Teytaud. 2016. Algorithm portfolios for noisy optimization. *Annals of Mathematics and Artificial Intelligence* 76, 1-2 (2016), 143–172.

[6] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (2008), 47–62.

[7] David B Fogel and Evolutionary Computation. 1995. Toward a New Philosophy of Machine Intelligence. *IEEE Evolutionary Computation* (1995).

[8] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. 2018. Noisy networks for exploration. *International Conference on Learning Representations* (2018).

[9] Scott Fujimoto, Herke van Hoof, and Dave Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*.

[10] Matteo Gagliolo and Jürgen Schmidhuber. 2006. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence* 47, 3-4 (2006), 295–328.

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

[12] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[13] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. 2018. Distributed prioritized experience replay. In *International Conference on Learning Representations*.

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

[15] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. 2019. Collaborative Evolutionary Reinforcement Learning. In *International Conference on Machine Learning*.

[16] Shauharda Khadka and Kagan Tumer. 2018. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*. 1188–1200.

[17] Michail G Lagoudakis and Michael L Littman. 2000. Algorithm Selection using Reinforcement Learning.. In *ICML*. Citeseer, 511–518.

[18] Romain Laroche and Raphaël Feraud. 2018. Reinforcement Learning Algorithm Selection. In *International Conference on Learning Representations*.

[19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*.

[20] Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8, 3-4 (1992), 293–321.

[21] Benno Lüders, Mikkel Schläger, Aleksandra Korach, and Sebastian Risi. 2017. Continual and one-shot learning through neural networks with dynamic external memory. In *European Conference on the Applications of Evolutionary Computation*. Springer, 886–901.

[22] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[24] Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. 2017. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

[25] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 16–17.

[26] Aloïs Pourchot and Olivier Sigaud. 2018. CEM-RL: Combining evolutionary and gradient-based methods for policy search. *International Conference on Learning Representations* (2018).

[27] John R Rice. 1976. The algorithm selection problem. In *Advances in computers*. Vol. 15. Elsevier, 65–118.

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. In *International Conference on Learning Representations*.

[29] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. 1889–1897.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[31] Daniel Shaefer and Scott Ferguson. 2013. Using a goal-switching selection operator in multi-objective genetic algorithm optimization problems. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection.

[32] Kate A Smith-Miles. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)* 41, 1 (2009), 6.

[33] Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2019. Learning Variable Ordering Heuristics for Solving Constraint Satisfaction Problems. *arXiv preprint arXiv:1912.10762* (2019).

[34] William M Spears, Kenneth A De Jong, Thomas Bäck, David B Fogel, and Hugo De Garis. 1993. An overview of evolutionary computation. In *European Conference on Machine Learning*. Springer, 442–459.

[35] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[36] Richard S Sutton and Andrew G Barto. 2011. Reinforcement learning: An introduction. (2011).

[37] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. 2017. Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*. 2753–2762.

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

[39] Pengfei Wei, Yiping Ke, and Chi Keong Goh. 2016. Deep nonlinear feature coding for unsupervised domain adaptation.. In *IJCAI*. 2189–2195.

[40] Zhizheng Zhang, Jiale Chen, Zhibo Chen, and Weiping Li. 2019. Asynchronous Episodic Deep Deterministic Policy Gradient: Towards Continuous Control in Computationally Complex Environments. *arXiv preprint arXiv:1903.00827* (2019).