# Double Buffers CEM-TD3: More Efficient Evolution and Richer Exploration

**Sheng Zhu**[1,2], **Chun Shen**[1,2,3], **Shuai Lü**[1,2,3,*], **Junhong Wu**[1,2], **Daolong An**[1,3]

[1]Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, China
[2]College of Software, Jilin University, Changchun 130012, China
[3]College of Computer Science and Technology, Jilin University, China
{lus, shenchun}@jlu.edu.cn, {zhusheng20, chwu22, andl22}@mails.jlu.edu.cn

## Abstract

CEM-TD3 is a combination scheme using the simple cross-entropy method (CEM) and Twin Delayed Deep Deterministic policy gradient (TD3), and it achieves a satisfactory trade-off between performance and sample efficiency. However, we find that CEM-TD3 cannot fully address the low efficiency of policy search caused by CEM, and the policy gradient learning introduced by TD3 will weaken the diversity of individuals in the population. In this paper, we propose Double Buffers CEM-TD3 (DBCEM-TD3) that optimizes both CEM and TD3. For CEM, DBCEM-TD3 maintains an actor buffer to store the population required for evolution. In each iteration, it only needs to generate a small number of actors to replace the poor actors in the actor buffer to achieve more efficient evolution. The fitness of individuals in the actor buffer decreases exponentially with time, which can avoid premature convergence of the mean actor. For TD3, DBCEM-TD3 maintains a critic buffer with the same number of critics as the number of actors generated in each iteration, and each critic is trained independently by sampling from the shared replay buffer. In each iteration, each newly generated actor uses different critics to guide learning. This ensures more diverse behaviors among the learned actors, enabling richer experiences to be collected during the evaluation phase. We conduct experimental evaluations on five continuous control tasks provided by OpenAI Gym. DBCEM-TD3 outperforms CEM-TD3, TD3, and other classic off-policy reinforcement learning algorithms in terms of performance and sample efficiency.

## Introduction

Deep reinforcement learning (DRL) (Silver et al. 2014; Schulman et al. 2017; Haarnoja et al. 2018; Fujimoto, van Hoof, and Meger 2018) is the combination of deep neural network and reinforcement learning. DRL uses deep neural network as non-linear function approximator, which has stronger representation capability and is widely used in control tasks (Mnih et al. 2015, 2016; Lillicrap et al. 2016) . The goal of DRL is to find the optimal parameter settings from the family of functions that can maximize the expected cumulative reward. Although deep neural network can solve complex tasks, the large parameter space makes it more difficult for DRL to search for the optimal parameters. In addition, deep neural network is sensitive to hyperparameters (Henderson et al. 2018) and the training process is usually highly unstable (Fujimoto, van Hoof, and Meger 2018), which may cause learning policy to fluctuate or even not converge during the optimization process.

Evolutionary method (Salimans et al. 2017; Fogel 2006; Aulig and Olhofer 2016) seeks the optimal parameter settings by simulating the evolution process in nature. As an extensible alternative to reinforcement learning, evolutionary method is simple to implement and can be used for parallel searching. Evolutionary Reinforcement Learning (ERL) (Khadka and Tumer 2018) combines Evolutionary Algorithm (EA) with Reinforcement Learning (RL). Specifically, during RL agent training, EA provides diverse experiences. In the evolutionary stage, RL agents provide individuals with gradient information to EA. ERL combines the advantages of EA and RL, and its performance is better than EA and RL. Collaborative Evolutionary Reinforcement Learning (CERL) (Khadka et al. 2019) introduces a set of actors based on policy gradient learning, which exchange information with the population to enhance policy search capabilities. Proximal Distilled Evolutionary Reinforcement Learning (PDERL) (Bodnar, Day, and Li 2020) indicates that ERL does not fully solve the scalability problem of EA, and optimizes the crossover and mutation operators in ERL, further improving the performance of evolutionary reinforcement learning algorithms. However, whether it is ERL or PDERL, the combination scheme cannot improve the search efficiency of EA.

CEM-TD3 combines the simple cross-entropy method (CEM) (Duan et al. 2016) and Twin Delayed Deep Deterministic policy gradient (TD3) (Fujimoto, van Hoof, and Meger 2018), and it achieves a satisfactory trade-off between performance and sample efficiency. However, we find that there is room for further optimization in CEM-TD3. On the one hand, CEM only uses half of the actors to evolve, and the other half of the actors are eliminated, which will lead to a waste of a large number of actors and low evolutionary efficiency. On the other hand, half of the actors generated by CEM are trained based on the same critic in TD3. Although this will provide policy gradient information to the population, it leads to similar behaviors among these actors, thereby reducing population diversity. This paper optimizes

*Corresponding author

CEM-TD3 to alleviate the above issues, and the contributions of this paper are as follows:

- We indicate that in CEM-TD3, the policy search efficiency of CEM is low. We optimize the evolution process of CEM by introducing an actor buffer and proposing a more effective policy search approach.
- We indicate that the gradient learning guidance for population actors in CEM-TD3 weakens the diversity of population. We maintain a critic buffer and use the bootstrap method to guide actors in the population, which can increase the diversity and collect more valuable experiences.
- We propose Double Buffers CEM-TD3 (DBCEM-TD3) based on these improvement measures. Through experiments, we show that DBCEM-TD3 significantly outperforms CEM-TD3. Compared with evolutionary reinforcement learning methods, DBCEM-TD3 can better balance performance and sample efficiency. In addition, we comprehensively compare DBCEM-TD3 and CEM-TD3 from five perspectives: performance, time cost, evolutionary efficiency, sample efficiency, and population diversity.

## Related Work

EA is a black box search algorithm that simulates the biological evolution process in nature. (Salimans et al. 2017) uses EA to solve the RL problem. RL (Silver et al. 2014; Schulman et al. 2017; Haarnoja et al. 2018; Fujimoto, van Hoof, and Meger 2018; Mnih et al. 2013) is optimization algorithm based on gradient information. The research direction of this paper is the combination of EA and RL. (Khadka and Tumer 2018) first proposes the ERL framework, which combines EA with RL. The framework constructs a population for evolution and an actor-critic agent based on Deep Deterministic Policy Gradient (DDPG). DDPG regularly injects actors with gradient information into the population, proposing gradient information for the evolution of the population. The experiences collected by the population during evaluation are filled into the replay buffer, providing diverse experiences for DDPG training. The framework successfully integrates EA and RL and achieves higher performance. (Bodnar, Day, and Li 2020) proposes Proximal Distilled Evolutionary Reinforcement Learning (PDERL) based on ERL. They indicate catastrophic forgetting in the evolution process of neural network, and optimize the crossover and mutation operators in the evolution process of ERL. (Khadka et al. 2019) proposes Collaborative Evolutionary Reinforcement Learning (CERL) based on ERL. CERL uses a combination of learners with different time horizons for learning and injects them into the population, providing individuals with richer gradient information for the population. (Pourchot and Sigaud 2019) proposes a different combination scheme named CEM-TD3 using the Cross-Entropy Method (CEM) and Twin Delayed Deep Deterministic policy gradient (TD3). Compared with the ERL framework, CEM-TD3 can greatly improve the policy search efficiency of evolutionary algorithms.

ERL, PDERL, CERL, and CEM-TD3 all enhance the exploration ability of RL agents through experiences provided by populations. (Lyu et al. 2022; Jung, Park, and Sung 2020; Pan, Cai, and Huang 2020) use multiple actors to improve the exploration. (Plappert et al. 2018) adds noise directly to the policy network, which improves the ability of consistency exploration through parameter space noise. Moreover, the count-based (Bellemare et al. 2016; Burda et al. 2019) and curiosity-driven (Stadie, Levine, and Abbeel 2015; Ostrovski et al. 2017) exploration methods guide the policy to explore more effectively through additional rewards. CEM-TD3 has higher search efficiency compared to ERL, PDERL, and CERL. Our work further improves the search efficiency of CEM-TD3 and enhances the diversity of the population weakened by policy gradient learning in CEM-TD3.

## Background

DDPG and TD3 are two deterministic policy-based, actor-critic (Konda and Tsitsiklis 1999) algorithms. They use deep neural network as function approximator. Unfortunately, DDPG has the same overestimation problem as Deep $Q$-Leaning Network (DQN) (Mnih et al. 2013). Although Double DQN (DDQN) (van Hasselt, Guez, and Silver 2016) can alleviate the overestimation problem of DQN, (Fujimoto, van Hoof, and Meger 2018) indicates that it cannot be directly applied to DDPG and proposes TD3 to alleviate the overestimation problem of DDPG. Specifically, TD3 introduces two critics and reduces overestimation bias by learning the lower bounds of the two critics. In addition, TD3 adds noise to the target action to smooth the value function and reduces error accumulation through policy delayed updates.

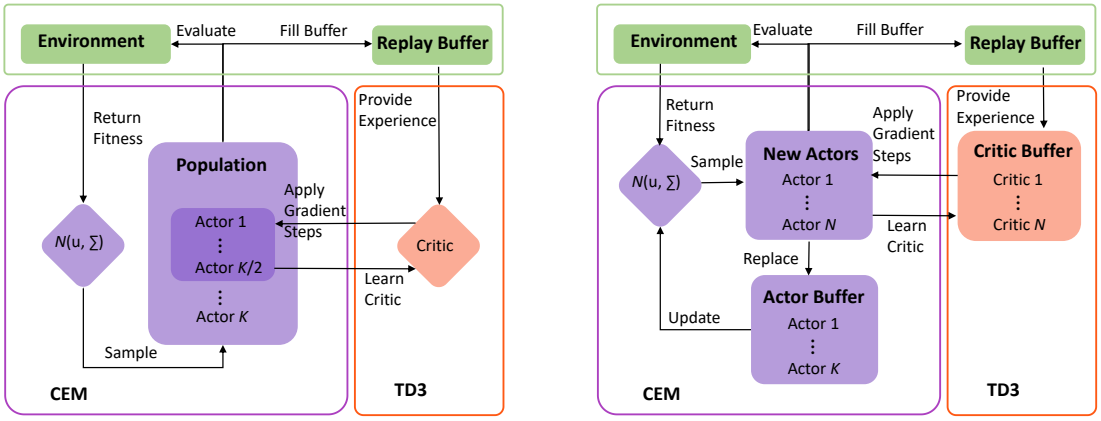
CEM is an estimation of distribution algorithms. CEM maintains a distribution of mean $\mu$ and covariance matrix $\Sigma$ and keeps updating the distribution iteratively. In each iteration, CEM samples $K$ individuals from the distribution $\mathcal{N}(\mu, \Sigma)$ and then evaluates the fitness of these $K$ individuals based on optimization objectives. Finally, CEM selects the $K_e$ individuals with the highest fitness and uses the following formula to update the distribution $\mathcal{N}(\mu, \Sigma)$:

$$\mu_{new} = \sum_{i=1}^{K_e} \lambda_i z_i \tag{1}$$

$$\Sigma_{new} = \sum_{i=1}^{K_e} \lambda_i (z_i - \mu_{old})^2 + \epsilon \mathcal{I} \tag{2}$$

where $z_i$ represents the individual with the $i$-th highest fitness, $\epsilon$ is the noise injected into the covariance matrix $\Sigma$, with an initial value of $\epsilon_{init}$ and exponentially reduced by $\epsilon = \tau_{cem}\epsilon_{init} + (1 - \tau_{cem})\epsilon_{end}$, which keeps unchanged after lowering to $\epsilon_{end}$. The purpose of $\epsilon\mathcal{I}$ is to prevent premature convergence caused by the original covariance being too small.

CEM-TD3 combines the above CEM and TD3, and its structure is shown in Figure 1(a). The core of CEM-TD3 is to optimize the distribution of policy based on CEM and




(a) CEM-TD3 (Pourchot and Sigaud 2019) (b) DBCEM-TD3

Figure 1: CEM-TD3 and DBCEM-TD3.

guide the learning of individuals in the population based on TD3 during the iteration process. Specifically, in each iteration, CEM generates a population of $K$ actors based on the current distribution $\mathcal{N}(\pi_\mu, \Sigma)$. Next, randomly select half of the actors to directly evaluate their fitness in the environment, where fitness is the cumulative reward of the actors interacting in the environment for one episode. The actor in the other half used to be form TD3 with the same critic, and optimize these actors and the critic. The optimized actors evaluate their fitness in the environment. Finally, half of the actors with higher fitness are used to update the distribution $\mathcal{N}(\pi_\mu, \Sigma)$ based on Formulas (1) and (2).

## Method

This section introduces our proposed DBCEM-TD3 framework and visualizes the problems of low evolutionary efficiency and low population diversity in CEM-TD3, as well as how DBCEM-TD3 improves these problems.

As shown in Figure 1(b), DBCEM-TD3 replaces the population in CEM-TD3 with two storage areas for actors, namely the new actors and the actor buffer. The size of the new actors is $N$. The size of the actor buffer is $K$. $N$ is less than $K$, and the size of the actor buffer is equal to the population size of CEM-TD3. The new actors are responsible for generating $N$ new individuals at the current iteration time. The actor buffer stores individuals generated in the past. Each new actor generated will determine whether to replace individuals in the actor buffer based on certain rules. The individual in the actor buffer is used to update the mean $\pi_\mu$ and covariance matrix $\Sigma$ of CEM, and the principle of updating is the same as CEM-TD3. In addition, DBCEM-TD3 maintains a critic buffer of size $N$. It is worth noting that the size of the critic buffer is equal to the size of the new actors, and each critic is matched with a new actor, optimized using TD3. Next, we will provide a detailed description of the implementation details involved in DBCEM-TD3 and how to improve CEM-TD3.

### Warming Up Stage

The mean actor of CEM is randomly initialized, and all critics in the critic buffer are also randomly initialized. The population size used for CEM evolution is usually 10, i.e. $K = 10$. It is worth noting that due to the empty actor buffer during initialization, the number of new actors generated in the first iteration is insufficient for the number of individuals required for CEM evolution. Therefore, before the first iteration, 10 actors are sampled based on the current mean $\pi_\mu$ and the current covariance matrix $\Sigma$, and evaluated in the environment to obtain the fitness of 10 actors ($f_i$ represents the fitness of the $i$-th actor). Fill these 10 actors into the actor buffer.

### CEM Stage of DBCEM-TD3

In iteration $t$, $N$ new actors are sampled based on the current mean $\pi_\mu$ and the current covariance matrix $\Sigma$. Each new actor will be matched with one critic in the critic buffer, and then TD3 will be used for policy optimization. Each trained new actor is evaluated in the environment to obtain its corresponding fitness. The evaluated experiences are filled into the replay buffer. When a new actor can be selected as an elite in the actor buffer (i.e. the fitness of new actors is higher than that of half of the actors in the actor buffer), the new actor will replace the actor with the lowest fitness in the actor buffer. This is reasonable because CEM selects $K/2$ elite actors from the actor buffer to update $\pi_\mu$ and $\Sigma$, and the new actor can only affect the distribution of CEM by being selected as elites.

If in a certain iteration, none of the newly generated $N$ actors are selected as elites, then the actor buffer will still store past actors. In this case, we do not update the mean $\pi_\mu$ and covariance matrix $\Sigma$ (i.e. do not evolve). We clearly expect newly generated actors to be selected as elites, leading to the evolution of CEM. Because evolution means the emergence of higher performing actors, which will play a crucial role in searching for better policy. We run DBCEM-TD3 in Ant-v2 environment, and Figure 2 shows whether CEM has evolved during each iteration. In the early stages of training, due to

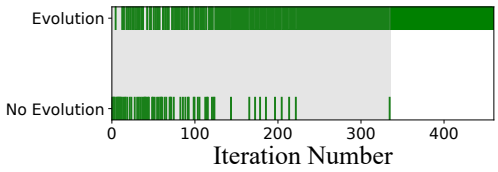
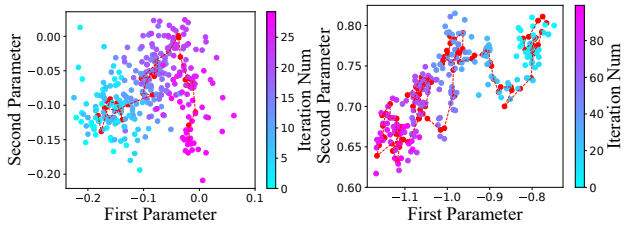
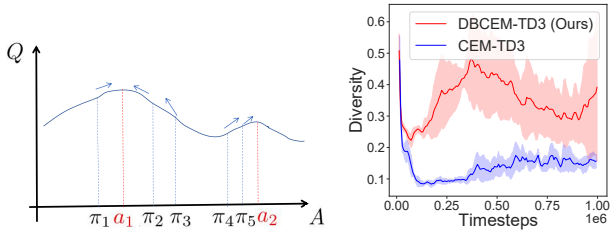



Figure 2: Visualization of evolution or not in each iteration.




(a) CEM-TD3 (b) DBCEM-TD3

Figure 3: Evolution of the first two parameters of the actors when learning with CEM-TD3 and DBCEM-TD3. Dots are sampled parameters of the population and red dashed lines represent the moving of the mean actor parameters.




(a) (b)

Figure 4: (a) The situation where different actors use the same critic for updates and (b) Comparison of differences in actor behavior between CEM-TD3 and DBCEM-TD3 populations.

the random initialization of mean $\pi_\mu$ and critics, the generated new actors are not easily selected as elites, resulting in slow evolution. But as the iteration progresses, new actors are more likely to be selected as elites, especially after the 336-th iteration, where new actors is selected as an elite in each iteration, leading to quick evolution. This is because on the one hand, CEM conducts policy searches in the direction of the elite, and on the other hand, critics in TD3 learn better value estimation, providing better guidance for new actors.

Whether a new actor is selected as an elite depends on whether the fitness of the new actor is higher than that of half of the actors in the actor buffer. We exponentially decay the fitness of all actors in the actor buffer during each iteration. Specifically, for a certain actor selected as an elite in the $t_i$-th iteration and filled in the actor buffer, $f_{t_i}$ represents the fitness of the actor when selected as an elite. At a certain iteration $t$ after the $t_i$-th iteration, if the actor still exists in the actor buffer, the fitness of the actor at the $t$-th iteration will become $\alpha^{(t-t_i)} f_{t_i}$, where $\alpha$ is the decay scaling factor that controls the degree of fitness decay. If an old actor is stored in the actor buffer for a longer time, its fitness will become lower. New actors will be more likely to replace it. The goal of fitness decay is to alleviate the premature convergence of actors and accelerate population search speed. If CEM generates a local optimal actor and stores it in the actor buffer, it is easy for the local optimal actor to be selected as an elite in the future, which will affect the updating of CEM distribution and slow down policy search speed. The fitness decay operation can alleviate this phenomenon.

CEM-TD3 evolves once by sampling $K$ actors each time, while our proposed DBCEM-TD3 evolves once by sampling $N$ actors each time. In the experiment, $K = 10$ and $N = 3$. In DBCEM-TD3, if there are elites among the three sampled new actors, it is beneficial for the policy search of CEM. Even if there is no elite, it only costs three actors. The original CEM samples 10 actors for optimization in each iteration, which will result in many actors being wasted. We will verify the existence of the above problems through experiments. Figure 3 shows the changes in population (dots) and mean actor (red line) during the iteration process of CEM-TD3 and DBCEM-TD3. Both CEM-TD3 and DBCEM-TD3 use the mean actor $\pi_\mu$ as the results of evolution, and the movement trajectory of the mean actor $\pi_\mu$ represents the direction of evolution. There are a large number of actors in Figure 3(a) that are far from the mean actor, meaning that these actors do not have a guiding effect on the moving direction of the mean actor, and these actors are somewhat wasteful. In Figure 3(b), only a small number of actors deviate from the moving direction of the mean actor, because DBCEM-TD3 only generates a small number of new actors for trial and error exploration. Compared to CEM-TD3, DBCEM-TD3 has higher search efficiency.

## TD3 Stage of DBCEM-TD3

In CEM-TD3, a critic guides half of the actors in the population to move. Although there are differences in the behavior of these actors, using the same $Q$ function to guide their learning will weaken their differences. As shown in Figure 4(a), the $Q$ function reaches its local maximum at action $a_1$ or $a_2$, and the 5 actors (represented by $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$) are trained based on the $Q$ function. These 5 actors will move towards the direction of the local maximum $Q$. Specifically, $\pi_1$, $\pi_2$ and $\pi_3$ will move towards action $a_1$, while $\pi_4$ and $\pi_5$ will move towards action $a_2$. Finally, the actions performed by $\pi_1$, $\pi_2$ and $\pi_3$ are very similar (all close to $a_1$), while the actions performed by $\pi_4$ and $\pi_5$ are very similar (all close to $a_2$). After training, these 5 actors will lack diversity. Therefore, in CEM-TD3, although TD3 provides gradient information guidance to the actors in the population, it weakens the diversity of the population.

To alleviate this issue, DBCEM-TD3 maintains a critic buffer of size $N$. After generating $N$ new actors in CEM, the new actors will randomly match with the critics and learn according to TD3. On the one hand, since these critics independently sample experiences from the replay buffer, the differences between critics can lead to differences in the learning of new actors. On the other hand, this optimization

Algorithm 1: DBCEM-TD3

Initialize the hyper-parameters $\alpha$, $\tau_{CEM}$, $\sigma_{init}$, $\sigma_{end}$ and $pop\_size$ of CEM
Initialize the hyper-parameters $\tau$, $\gamma$, $lr_{actor}$, $lr_{critic}$ and $pop\_size$ of TD3
Initialize the mean actor $\pi_\mu$ and the covariance matrix $\Sigma$ of CEM
Initialize the critic buffer $\{Q_1, ..., Q_N\}$
Extract $K$ actors from $\mathcal{N}(\pi_\mu, \Sigma)$ and fill them in the actor buffer $\{\pi_1, ..., \pi_K\}$
Evaluate $K$ actors in the actor buffer to obtain fitness $\{f_1, ..., f_K\}$ and steps $total\_steps$
$actor\_steps$, $max\_steps = total\_steps$, 1000000
**while** $total\_steps < max\_steps$ **do**
  Extract $N$ new actors $\{\pi'_1, ..., \pi'_N\}$ from $\mathcal{N}(\pi_\mu, \Sigma)$
  **for** $i = 1$ to $N$ **do**
    Train $Q_i$ and $\pi'_i$ for $actor\_steps$ mini-batches
  **end for**
  Evaluate all new actors to obtain fitness $\{f'_1, ..., f'_N\}$ and steps $actor\_steps$
  $total\_steps = total\_steps + actor\_steps$
  The fitness decay of all actors in the actor buffer: $\{f_1, ..., f_K\} \leftarrow \{\alpha f_1, ..., \alpha f_K\}$
  **for** $i = 1$ to $N$ **do**
    Find the worst fitness $f_{low}$ in $\{f_1, ..., f_K\}$
    **if** $f'_i > f_{low}$ **then**
      $\pi_{low}$, $f_{low} \leftarrow \pi'_i$, $f'_i$
    **end if**
  **end for**
  **if** New actors selected as elites **then**
    Update $\pi_\mu$ and $\Sigma$ with the top half of the actor buffer based on Formulas (1) and (2)
  **end if**
**end while**

method is similar to Bootstrapped DQN (Osband et al. 2016) and can improve deep exploration capabilities.

To further verify that the critic buffer can enhance the diversity of actors learning in the population, we define the difference calculation formula between the actor $\pi_1$ and the actor $\pi_2$ as $Dis(\pi_1||\pi_2) = \frac{1}{M}\sum_{i=1}^{M}[\pi_1(s_i) - \pi_2(s_i)]^2$, where $M$ is the batch size sampled from the replay buffer. Figure 4(b) shows the differences in the behavior of two actors guided by critics during each iteration process. The results indicate that the differences between actors guided by critics in DBCEM-TD3 are consistently higher than those in CEM-TD3. Therefore, the population of DBCEM-TD3 can better maintain individual differences while utilizing gradient learning information. The pseudo code of DBCEM-TD3 is shown in Algorithm 1.

## Experiments

In this section, we will answer the following questions through experiments:

- Compared with reinforcement learning and evolutionary reinforcement learning, what is the effect of DBCEM-TD3? How to set hyperparameters and how does it affect performance?
- Will CEM-TD3 lead to a decrease in population diversity due to TD3 learning? And does our DBCEM-TD3 enhance individual diversity in the population?
- How much improvement does DBCEM-TD3 have in performance, sample efficiency, evolutionary efficiency, and population diversity compared to CEM-TD3? How much additional time does DBCEM-TD3 introduce compared to CEM-TD3?

### Experimental Setup

We conduct experiments on five continuous control tasks on MuJoCo (Todorov, Erez, and Tassa 2012) hosted on OpenAI Gym (Brockman et al. 2016). We replicate TD3[1], SAC[2], ERL[3], PDERL[4], CERL[5], and CEM-TD3[6] based on their open-source code. Our DBCEM-TD3 implementation is based on CEM-TD3, and the parameter settings are shown in Appendix. The performance of all curves is the average of 10 evaluations every 5000 steps. The shaded area of the curve corresponds to the standard deviation, and all curves are smoothed. Our code is available at https://github.com/ShengZhujli/DBCEMTD3.

### Performance Evaluation

Figure 5 shows the learning curve of our DBCEM-TD3 and the baselines on five tasks. The results indicate that compared to CEM-TD3, SAC and TD3, DBCEM-TD3 has significantly higher performance and faster learning speed in all testing tasks. Compared with ERL, PDERL and CERL, DBCEM-TD3 has significantly faster learning speed and higher performance on Ant-v2, HalfCheetah-v2, Hopper-v2, and Walker-v2 at 1 million timesteps. ERL, PDERL, and CERL are more focused on the evolutionary process, which will require a lot of evaluation, resulting in slow learning speed. DBCEM-TD3 and CEM-TD3 focus more on the process of RL, so they inherit the fast speed of RL. Due to the large fluctuation of rewards on Swimmer-v2, the evolutionary process focused approach can effectively search for the policy with higher global performance using episode reward, while the RL focused CEM-TD3 and DBCEM-TD3 can hardly search for the global optimal policy using single step bootstrap update in the RL process. Therefore, only the evolutionary RL algorithms can learn more than 300 returns on Swimmer-v2. Table 1 presents the final performance comparison of these algorithms at 1 million timesteps. As a result, DBCEM-TD3 learnes better performance and smaller variance policy than CEM-TD3 in all testing tasks.

### Hyperparameter Setting

Hyperparameter $N$ not only represents the number of generated new actors at each iteration, but also the number of

[1]https://github.com/sfujim/TD3

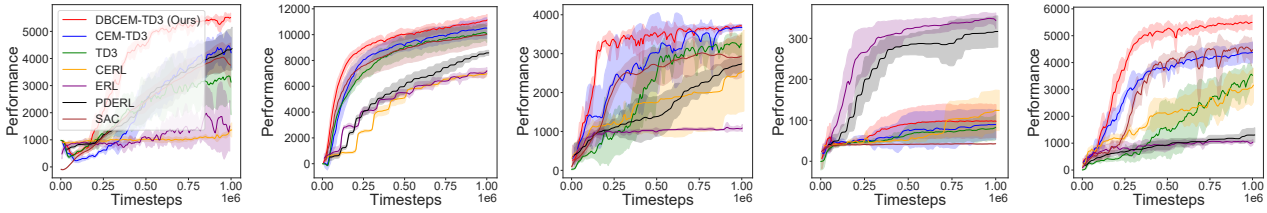
[2]https://github.com/openai/spinningup/tree/master/spinup/algos/pytorch/sac

[3]https://github.com/ShawK91/erl_paper_nips18

[4]https://github.com/crisbodnar/pderl

[5]https://github.com/intelai/cerl

[6]https://github.com/apourchot/CEM-RL

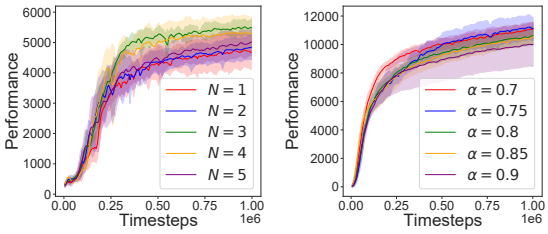



(a) Ant-v2 (b) HalfCheetah-v2 (c) Hopper-v2 (d) Swimmer-v2 (e) Walker2d-v2

Figure 5: Learning curves of DBCEM-TD3, CEM-TD3, TD3, CERL, ERL, PDERL and SAC.

| Environment | SAC | TD3 | ERL | PDERL | CERL | CEM-TD3 | DBCEM-TD3 |
|---|---|---|---|---|---|---|---|
| Ant-v2 | 3735.40 | 3773.16 | 1089.20 | 1496.77 | 3209.58 | 4381.43±395.45 | **5483.31±296.61** |
| HalfCheetah-v2 | 10024.72 | 10245.54 | 7216.24 | 8586.78 | 7284.29 | 10503.32±806.29 | **11174.98±422.71** |
| Hopper-v2 | 2943.659 | 2853.73 | 1117.34 | 2754.93 | 2576.11 | 3667.23±70.09 | **3698.08±35.48** |
| Swimmer-v2 | 74.05 | 87.24 | **342.53** | 316.81 | 124.20 | 89.75±36.24 | 97.17±40.52 |
| Walker2d-v2 | 4560.54 | 3989.87 | 1089.20 | 1496.77 | 3209.58 | 4381.43±395.45 | **5483.31±296.61** |

Table 1: Final performance comparison. Average return of 1M timesteps. ±corresponds to the standard deviation of 5 trials.




(a) $N$ (b) $\alpha$

Figure 6: Comparison performance with different hyperparameter settings in HalfCheetah-v2.

critics in the critic buffer. On the one hand, a small $N$ can accelerate the process of evolution, but the corresponding number of critics is small, and the individual diversity for policy gradient learning will be smaller. On the other hand, a large $N$ will slow down the speed of evolution, but an increase in the number of critics is beneficial for learning more diverse actors. An appropriate $N$ balances the speed of evolution and individual diversity of the population. Figure 6(a) shows the impact of different $N$ on performance. The results indicate that the performance is best when $N = 3$. For convenience, we set $N = 3$ for all environments. From the results in Figure 5 and Table 1, it can be seen that $N = 3$ has achieved good results.

Hyperparameter $\alpha$ controls the degree of decay of actor fitness in the actor buffer. The smaller $\alpha$, the faster the fitness of the actor in the actor buffer decays, and the easier it is to be replaced by new actors generated by CEM. New actors participate in population evolution and can explore new behaviors. Therefore, the larger $\alpha$ can be beneficial for population exploration and prevent premature convergence of actors to local optima. The larger $\alpha$, the higher the fitness of the new actor to replace the old actor in the actor buffer. Therefore, the evolution speed may slow down, but it is more conducive to the population's evolution towards better performance. Figure 6(b) shows the impact of different $\alpha$ on performance on HalfCheetah-v2. The performance is best when $\alpha = 0.7$. Compared to other values, when $\alpha = 0.7$, it is easier for the new actor to replace old one in the buffer, resulting in stronger exploration capability and faster learning speed in the early stage. Therefore, when $\alpha = 0.7$, the performance is significantly better than other values in the 0.5 million timesteps of training. Setting appropriate $\alpha = 0.7$ values for different tasks can improve learning speed and algorithm performance.

## Comparison of Population Diversity

In CEM-TD3, half of the actors in the population are trained based on the same critic, which will result in very similar behaviors among the trained actors. To verify this phenomenon, we visualize the state visitation distributions of these actors interacting with the environment 10000 times on Ant-v2. Figure 7(a) shows the state visitation distribution of the mean actor in CEM-TD3. Figure 7(b), (c) and (d) show the state visitation distributions of three trained actors generated based on mean actors in CEM-TD3. The results indicate that although there are differences in the state visitation distributions between the three trained actors and the mean actor, the state visitation distributions among these three trained actors are very similar. On the one hand, actors with similar behaviors have consistent evolutionary directions during the evolutionary process, which in fact leads to the waste of actors. Especially in the evolutionary process, the sample efficiency is low, and the similarity of such actors will further reduce the sample efficiency. On the other hand, these similar behaviors are stored in the replay buffer, which cannot bring richer experiences to TD3 learning.

In our DBCEM-TD3, new actors generated during each evolution are trained under the guidance of different critics

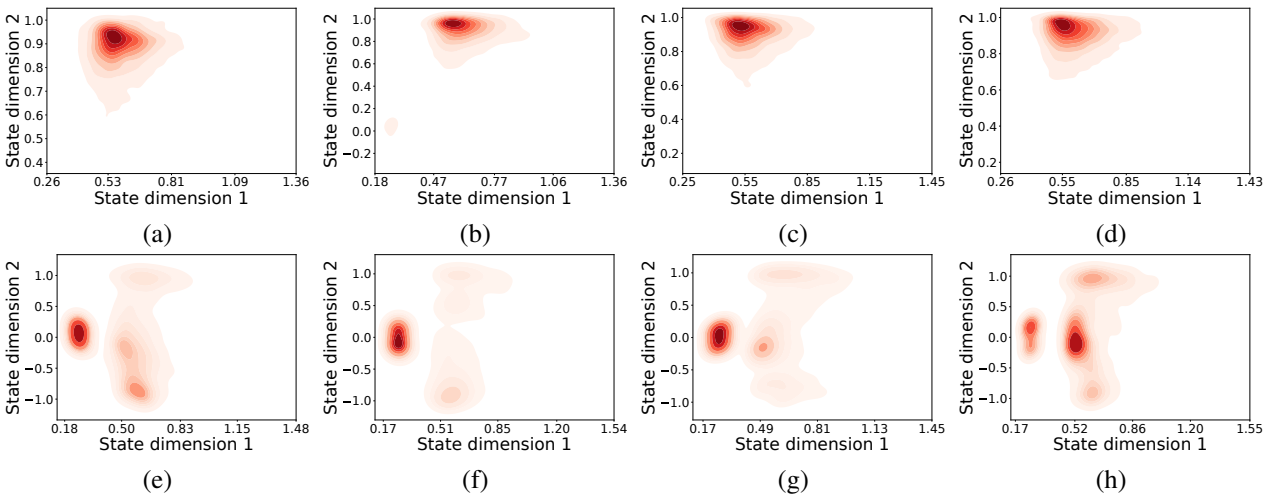



Figure 7: The state visitation distributions for different actors in Ant-v2. (a) Mean actor of CEM-TD3. (b) Actor 1 sampled by mean actor of CEM-TD3. (c) Actor 2 sampled by mean actor of CEM-TD3. (d) Actor 3 sampled by mean actor of CEM-TD3. (e) Mean actor of DBCEM-TD3. (f) Actor 1 sampled by mean actor of DBCEM-TD3. (g) Actor 2 sampled by mean actor of DBCEM-TD3. (h) Actor 3 sampled by mean actor of DBCEM-TD3.

in the critic buffer, which can not only enable the population to obtain policy gradient information, but also ensure the diversity of the populations. Figure 7(e) shows the state visitation distribution of the mean actor in DBCEM-TD3. Figure 7(f), (g) and (h) show the state visitation distributions of three trained actors generated based on mean actors in DBCEM-TD3. The results indicate that there are larger differences in the state visitation distributions among the three trained actors, which will be more conducive to finding the global optimal actor in evolution and providing richer experiences for TD3 learning.

## Comparison of Various Indicators

Table 2 shows the comparison of different indicators between CEM-TD3 and DBCEM-TD3. The final performance of DBCEM-TD3 at 1 million timesteps is improved by 25.09% compared to CEM-TD3. Time cost represents the time spent in training to 1 million timesteps. DBCEM-TD3 incurs an additional 12.50% time overhead compared to CEM-TD3. The evolutionary efficiency of DBCEM-TD3 is by 189.78% compared to CEM-TD3. Sample efficiency refers to the ratio of the number of interactions with the environment to the total number of interactions when reaching the highest performance of CEM-TD3. The sample efficiency of DBCEM-TD3 is improved by 55.61% compared to CEM-TD3. That is, DBCEM-TD3 interacts with the environment 443,900 times, achieving the performance of CEM-TD3 interacting with the environment 1 million times. The population diversity is calculated using $\frac{1}{T}\sum_{t=1}^{T}\{\frac{1}{M}\sum_{i=1}^{M}[\pi_1(s_i^t) - \pi_2(s_i^t)]^2\}$, where $t$ is the number of iterations, $M$ is the number of experiences sampled from the replay buffer, and $\pi_1$ and $\pi_2$ are two randomly selected actors from the population. The population diversity of DBCEM-TD3 is improved by 241.7% compared to CEM-TD3. DBCEM-TD3 outperforms CEM-TD3 in terms of performance, evolutionary efficiency, sample efficiency, and population diversity. In contrast, the additional time cost brought by DBCEM-TD3 is tolerable.

| Indicators | CEM-TD3 | DBCEM-TD3 |
|---|---|---|
| Final performance | 4383.43 | **5483.31** |
| Time cost (Minutes) | **184** | 207 |
| Evolutinary number | 137 | **397** |
| Sample efficiency | 100% | **155.61%** |
| Population diversity | 0.12 | **0.41** |

Table 2: Comparison of various indicators between DBCEM-TD3 and CEM-TD3 in Ant-v2.

## Conclusion

Our research direction is the combination of EA and RL. CEM-TD3 is a method with good sample efficiency and performance in this direction. However, we find that CEM-TD3 still has two issues. One is the slow evolution speed of CEM. Another is the low population diversity caused by TD3 learning. We propose DBCEM-TD3 to alleviate the above issues. For CEM, we introduce an actor buffer to store old actors, which can ensure the use of a small number of new actor for evolution. We verify that this is a more efficient way of evolution. While evolving faster, we perform exponential decay on the fitness of the actors in the actor buffer to prevent premature convergence of the mean actor. For TD3, we introduce a critic buffer to guide the learning of actors. We verify that this approach can provide actors with both gradient information and diversity to the population. The experimental results show that DBCEM-TD3 has significantly improved sample efficiency and performance compared to CEM-TD3. Finally, we validate the advantages of DBCEM-TD3.

## Acknowledgments

We sincerely thank the anonymous reviewers for their careful work and thoughtful suggestions, which have greatly improved this article. This work was supported by the Natural Science Research Foundation of Jilin Province of China under Grant Nos. 20220101106JC and YDZJ202201ZYTS423.

## References

Aulig, N.; and Olhofer, M. 2016. Evolutionary Computation for Topology Optimization of Mechanical Structures: An Overview of Representations. In *IEEE Congress on Evolutionary Computation (CEC 2016)*, 1948–1955.

Bellemare, M. G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying Count-Based Exploration and Intrinsic Motivation. In *Annual Conference on Neural Information Processing Systems (NIPS 2016)*, 1471–1479.

Bodnar, C.; Day, B.; and Li, P. 2020. Proximal Distilled Evolutionary Reinforcement Learning. In *AAAI Conference on Artificial Intelligence (AAAI 2020)*, 3283–3290.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. arXiv:1606.01540.

Burda, Y.; Edwards, H.; Storkey, A. J.; and Klimov, O. 2019. Exploration by Random Network Distillation. In *International Conference on Machine Learning (ICML 2019)*.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning (ICML 2016)*, 1329–1338.

Fogel, D. B. 2006. *Evolutionary Computation - Toward a New Philosophy of Machine Intelligence*. Wiley-VCH. ISBN 978-0-471-66951-7.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning (ICML 2018)*, 1582–1591.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning (ICML 2018)*, 1856–1865.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2018. Deep Reinforcement Learning That Matters. In *AAAI Conference on Artificial Intelligence (AAAI 2018)*, 3207–3214.

Jung, W.; Park, G.; and Sung, Y. 2020. Population-Guided Parallel Policy Search for Reinforcement Learning. In *International Conference on Learning Representations (ICLR 2020)*.

Khadka, S.; Majumdar, S.; Nassar, T.; Dwiel, Z.; Tumer, E.; Miret, S.; Liu, Y.; and Tumer, K. 2019. Collaborative Evolutionary Reinforcement Learning. In *International Conference on Machine Learning (ICML 2019)*, 3341–3350.

Khadka, S.; and Tumer, K. 2018. Evolution-Guided Policy Gradient in Reinforcement Learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS 2018)*, 1196–1208.

Konda, V. R.; and Tsitsiklis, J. N. 1999. Actor-Critic Algorithms. In *Annual Conference on Neural Information Processing Systems (NIPS 1999)*, 1008–1014.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous Control with Deep Reinforcement Learning. In *International Conference on Learning Representations (ICLR 2016)*.

Lyu, J.; Ma, X.; Yan, J.; and Li, X. 2022. Efficient Continuous Control with Double Actors and Regularized Critics. In *AAAI Conference on Artificial Intelligence (AAAI 2022)*, 7655–7663.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML 2016)*, 1928–1937.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. In *Annual Conference on Neural Information Processing Systems (NIPS 2013)*, 201–220.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540): 529–533.

Osband, I.; Blundell, C.; Pritzel, A.; and Roy, B. V. 2016. Deep Exploration via Bootstrapped DQN. In *Annual Conference on Neural Information Processing Systems (NIPS 2016)*, 4026–4034.

Ostrovski, G.; Bellemare, M. G.; van den Oord, A.; and Munos, R. 2017. Count-Based Exploration with Neural Density Models. In *International Conference on Machine Learning (ICML 2017)*, 2778–2787.

Pan, L.; Cai, Q.; and Huang, L. 2020. Softmax Deep Double Deterministic Policy Gradients. In *Annual Conference on Neural Information Processing Systems (NeurIPS 2020)*.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2018. Parameter Space Noise for Exploration. In *International Conference on Learning Representations (ICLR 2018)*.

Pourchot, A.; and Sigaud, O. 2019. CEM-RL: Combining Evolutionary and Gradient-based Methods for Policy Search. In *International Conference on Learning Representations (ICLR 2019)*.

Salimans, T.; Ho, J.; Chen, X.; and Sutskever, I. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. A. 2014. Deterministic Policy Gradient Algorithms. In *International Conference on Machine Learning (ICML 2014)*, 387–395.

Stadie, B. C.; Levine, S.; and Abbeel, P. 2015. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models. arXiv:1507.00814.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A Physics Engine for Model-based Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012)*, 5026–5033.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-learning. In *AAAI Conference on Artificial Intelligence (AAAI 2016)*, 2094–2100.