Hindawi
Computational Intelligence and Neuroscience
Volume 2021, Article ID 5300189, 11 pages
https://doi.org/10.1155/2021/5300189

Research Article

Diversity Evolutionary Policy Deep Reinforcement Learning

Jian Liu [1,2] and Liming Feng [1,2]

[1] *School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China*
[2] *Engineering Research Center of Intelligent Control for Underground Space, Ministry of Education, China University of Mining and Technology, Xuzhou 221116, China*

Correspondence should be addressed to Jian Liu; liujiansqjxt@126.com

Received 19 June 2021; Revised 10 July 2021; Accepted 19 July 2021; Published 4 August 2021

Academic Editor: Nian Zhang

The reinforcement learning algorithms based on policy gradient may fall into local optimal due to gradient disappearance during the update process, which in turn affects the exploration ability of the reinforcement learning agent. In order to solve the above problem, in this paper, the cross-entropy method (CEM) in evolution policy, maximum mean difference (MMD), and twin delayed deep deterministic policy gradient algorithm (TD3) are combined to propose a diversity evolutionary policy deep reinforcement learning (DEPRL) algorithm. By using the maximum mean discrepancy as a measure of the distance between different policies, some of the policies in the population maximize the distance between them and the previous generation of policies while maximizing the cumulative return during the gradient update. Furthermore, combining the cumulative returns and the distance between policies as the fitness of the population encourages more diversity in the offspring policies, which in turn can reduce the risk of falling into local optimal due to the disappearance of the gradient. The results in the MuJoCo test environment show that DEPRL has achieved excellent performance on continuous control tasks; especially in the Ant-v2 environment, the return of DEPRL ultimately achieved a nearly 20% improvement compared to TD3.

1. Introduction

Reinforcement learning [1, 2], as an important branch of machine learning [3, 4], has always been a research hotspot. Reinforcement learning constantly improves its policy by interacting with the actual environment, so that the policy can get the maximum cumulative return in the current environment. In recent years, deep learning has exerted more and more influence on various research fields. The combination of deep learning and reinforcement learning produces a variety of deep reinforcement learning algorithms. Deep reinforcement learning can be divided into three types: value-based deep reinforcement learning [5–7], policy-based deep reinforcement learning [8], and deep reinforcement learning based on actor-critic structure [9–11].

Value-based deep reinforcement learning methods estimate the value function through a neural network and use the value function output by the neural network to guide the agent to choose policies, such as deep Q network (DQN) algorithm [12]. Policy-based deep reinforcement learning methods can parameterize policies and achieve policy optimization through learning parameters, so that the agent can obtain the largest cumulative return, such as deterministic policy gradient (DPG) algorithm [5]. This type of algorithm has good performance when dealing with high-dimensional continuous space problems, but it is easy to cause gradient disappearance in the process of policy update and then fall into the local optimal solution problem [8]. Deep reinforcement learning methods based on actor-critic structure combine value-based and policy-based methods to learn policies while fitting value functions, such as deep deterministic policy gradient (DDPG) algorithm. Actor network parameters are trained according to the value function output by the critic network, and the critic network parameters are updated in a single step using the time difference (TD) method. Although the actor-critic-based methods have the advantages of both value-based and policy-based methods, they also inherit the shortcomings of the policy gradient algorithm; that is, the policy update falls into a local optimal solution due to the disappearance of the gradient.

The DDPG algorithm combines the ideas of DQN [12] and DPG [5] to solve tasks under continuous action. As an off-policy actor-critic algorithm, DDPG can be trained with historical data through experience playback pool, which greatly improves the utilization of samples and achieves better results in continuous action tasks. Subsequently, inspired by double DQN [13], twin delayed deep deterministic policy gradient algorithm (TD3) [10] on the basis of DDPG simultaneously uses two critic networks to fit the state action value function. And it takes the minimum value of the two target network outputs as the final estimate. TD3 solves the problem of overestimation of the DDPG median function and improves the stability of the agent. However, since DDPG and TD3 both use a similar way to the policy-based algorithms when updating the policy, they also rely on the gradient information for updating policy, which undoubtedly suffers from the vanishing gradient problem during the update process. By adding a small amount of random noise to the policy output by the neural network, the influence of the disappearance of the gradient on the policy update can be alleviated to a certain extent. For example, NoisyNets [14] enhance the exploration ability of the algorithm by directly adding random noise to the parameters of the neural network. However, since the influence of random noise on the policy is random and nondirectional, the effect of this method is limited. The combination of policy gradient and deep learning can be applied to complex and challenging tasks such as game simulation [15], robot control [16], and dialogue system [17]. However, when the policy gradient methods are applied to the continuous control filed, there still exists a basic problem, that is, the local optimal problem caused by the disappearance of gradient in the updating process. Tessler et al. [8] put forward that the generation model can be used to learn policies. In this way, although local optimal problem can be avoided, the difficulty of algorithm training is increased.

Evolutionary policy has been used as a nongradient optimization algorithm for decades and performs well in some reinforcement learning benchmark environments. Compared with gradient optimization, the evolution policy is simpler to implement, uses fewer hyperparameters, does not require gradient information, is easier to expand in a distributed environment, and is less affected by sparse rewards. Wierstra et al. [18] proposed Natural Evolution Policies (NES), which optimizes the policy by searching for the distribution of parameters and uses natural gradients to update the distribution in the direction of higher fitness. Inspired by the NES, Tim et al. [19] used the NES as a nongradient black box optimizer to find the optimal policy parameters. Khadka and Tumer [20] proposed evolutionary reinforcement learning (ERL) by effectively combining the evolutionary algorithm based on population with DDPG. Based on ERL, Pourchot and Sigaud [21] combined the cross-entropy method (CEM) with reinforcement learning and proposed CEM-RL method, which further improved the performance of the algorithm.

At present, most of the algorithms that combine reinforcement learning and evolutionary policy only make use of the cumulative return information of policies in each generation population but do not make full use of the distance information of policies between different generations. Effectively increasing the distance between policies of different generations is conducive to the generation of diverse policies for future generations and can improve the exploration of the environment by the reinforcement learning agent. Simultaneously, compared with the single policy, the diverse policies can effectively reduce the risk of falling into the local optimal solution in the updating process. Therefore, in this paper, a diversity evolutionary policy deep reinforcement learning (DEPRL) algorithm is proposed. DEPRL uses maximum mean discrepancy (MMD) to measure the distance between different policies. In the contemporary population, some policies maximize the cumulative return while maximizing the distance from the previous generation policy during the gradient update process. In the process of population evolution, the distance information and cumulative return of the policy are taken as the fitness of the population. The difference between the new generation policy and the previous generation policy is enlarged on the basis of ensuring the higher cumulative return of the new generation policy. By diversifying the policies in the population, DEPRL reduces the risk that the algorithm will fall into local optimum due to the disappearance of gradient in the process of updating and improves the exploration efficiency of agents. Finally, the effectiveness of DEPRL in continuous action task is verified by MuJoCo simulation environment.

The remainder of this paper is organized as follows. The next section describes the related works of DEPRL method. Section 3 represents the framework and details of DEPRL method. Then, in Section 4, a series of comparison experiments on MuJoCo test environment are conducted. The final section provides our concluding remarks and points out our future work orientation.

2. Related Works

2.1. Markov Decision Process (MDP). In reinforcement learning, the interaction process between reinforcement learning agents and the environment can be represented by Markov decision process (MDP). MDP can be represented by a tuple $M = (S, A, R, P^t, \gamma)$, where S is the state space, A is the action space, R is the reward function, P^t is the state transition probability, and $\gamma \in [0 \sim 1]$ is the discount factor. When the agent interacts with the environment, the way of choosing an action is called an action policy. Generally, the action policy can be a random policy or a deterministic policy. The random policy π is a probability value, which represents the probability that the agent chooses different actions from the action space in the state S, and the deterministic policy π_η represents the choice of a certain action in the state S. In each time step, the agent observes the current state $s_t \in S$ according to the environment and chooses action $a_t \sim \pi(s_t)$ according to the policy to get the reward $r_t = r(s_t, a_t)$ of the environment feedback.

Subsequently, the agent enters the next state according to the state transition probability P^t. The goal of reinforcement learning is to train the agent so that the agent finds an optimal policy π^* that can obtain the largest cumulative return.

2.2. Cross-Entropy Method (CEM). Evolutionary algorithms update the population by managing a finite number of individuals and generating new individuals near the previous elite sample. Some evolutionary algorithms are temporary optimization methods based on heuristics, such as genetic algorithm (GA) [22]. And the other part is based on the distribution algorithm that estimates the elite sample, such as estimation of distribution algorithms (EDA) [23, 24]. Cross-entropy method (CEM) is a simple EDA algorithm. Suppose that the total number of individuals in the population is K, where the total number of elite individuals is fixed at a certain value K_e, which is usually set to half of the total number of individuals in the population. After evaluating all the individuals in the population, the first K_e outstanding individuals are used to calculate the new mean and variance of the population. Then, additional variance is added to prevent premature convergence, and the next generation is sampled from the new population. A new distribution is obtained by adding Gaussian noise ε around the average value μ of the distribution, so that each individual $(x_i)_{i=1,\ldots,K}$ is sampled from this new distribution, that is, $x_i \sim \mathcal{N}(\mu, \Sigma)$, where Σ represents the current covariance matrix. By calculating the fitness of these newly generated individuals related to specific problems, CEM uses the best performing K_e individuals $(z_i)_{i=1,\ldots,K_e}$ to update the distribution parameters.

2.3. Neural Networks. In recent years, many neural networks, such as extreme learning machine (ELM) [25], probabilistic neural network (PNN) [26], and deep neural networks (DNN) [27], have been proposed and applied in many research fields. For example, Yi et al. [26] proposed a self-adaptive probabilistic neural network (SaPNN) method for transformer fault diagnosis problem. SaPNN can select the best spread self-adaptively all the time and always get the best prediction accuracy. To improve the accuracy and usefulness of target threat assessment in the aerial combat, Wang et al. proposed Elman-AdaBoost strong predictor [28] and multiple wavelet function wavelet neural networks (MWFWNN) [29] to solve threat assessment. Elman-AdaBoost strong predictor uses the Elman neural network as a weak predictor and obtains a strong predictor composed of multiple Elman neural network weak predictors through the Elman-AdaBoost algorithm. In [29], a wavelet mother function selection algorithm was proposed with minimum mean squared error and used to construct MWFWNN network. Cui et al. [30] proposed a novel method that used convolutional neural network (CNN) to improve the detection of malware variants. They converted the malicious code into grayscale images and used CNN to identify and classify the images.

Neural networks can also be applied to reinforcement learning. Traditional reinforcement learning is limited to small action space and sample space, which are generally discrete. However, more complex and more realistic tasks often have a large state space and continuous action space. When the input data is image or sound, it usually has a very high dimension, which is difficult for traditional reinforcement learning to deal with. Deep reinforcement learning is to combine the high-dimensional input of deep neural networks with reinforcement learning. Deep Q network (DQN) [12] can be regarded as the beginning of the successful combination of the two. It uses a deep network to represent the value function. Based on Q-learning in reinforcement learning, it provides target values for the deep network and constantly updates the network until convergence. After that, many deep reinforcement learning algorithms have been proposed, such as double DQN [13], DPG [5], and TD3 [10].

2.4. Twin Delayed Deep Deterministic Policy Gradient Algorithm (TD3). Both DDPG and TD3 are off-policy reinforcement learning algorithms based on the actor-critic structure. DDPG is easy to cause the problem of overestimation of value function, which affects the stability of algorithm. To mitigate the negative effects of overestimation, TD3 uses both critic networks to estimate the state action values and takes the minimum value of the two target network outputs as the final estimate.

In order to make the parameters of actor and critic networks updated stably, TD3 makes the updating frequency of network parameters of actor network lower than that of critic network during the training process. TD3 also adds random noise to the action output by the target policy, which not only improves the agent's exploration ability, but also fits the state action value of a small area around the target action. TD3 makes the value function learned by critic network smoother in the action dimension. Since the update direction of actor network parameters is affected by the value function learned from the critic network, the policy learned from actor network also tends to be smoother in the action dimension. By adding random noise, TD3 improves the stability of the agent during training process. The calculation formula of the action value of the target state in TD3 is as follows:

$$\begin{aligned} y(r, s') &= r + \gamma \min_{i=1,2} Q_{\phi_i}(s', \pi_\theta(s') + \varepsilon), \\ \varepsilon &\sim \operatorname{clip}(\mathcal{N}(0, \sigma), -c, c). \end{aligned} \tag{1}$$

3. Methods

3.1. Diversity Evolutionary Policy Deep Reinforcement Learning (DEPRL). The objective function of DEPRL mainly includes the objective function of critic network and actor network. To mitigate the impact of overestimation of the value function, critic network takes the minimum value of the two target network outputs to calculate the final target value. Assuming that θ_1 and θ_2 represent the estimated network parameters of the two critic networks, $\theta_{targ,1}$ and $\theta_{targ,2}$ represent target network parameters of the two critic networks. Then, the update process of the critic networks in DEPRL is shown in Figure 1. The target value of state action under time steps t is

$$Y(s_t, a_t) = r + \gamma \min_{i=1,2} Q_{\theta_{targ,i}}\left(s_{t+1}, \pi_\phi(s_{t+1})\right), \tag{2}$$

where r is the reward to the environment, $Q_{\theta_{targ,i}}(s_{t+1}, \pi_\phi(s_{t+1}))$ represents the target network output value of the i-th critic network, ϕ represents the network parameters of the actor network, and γ is the discount factor. Assume that $Q_{\theta_i}(s_t, a_t)$ represents the estimated value output by the i-th estimation network under the number of time steps t, and then the objective function of critic network can be written as

$$J_Q(\theta_i) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}\left(Q_{\theta_i}(s_t, a_t) - Y(s_t, a_t)\right)^2\right]. \tag{3}$$

Therefore, the estimated network parameters θ_1 and θ_2 can minimize the objective function $J_Q(\theta_i)$ through gradient descent. That is, gradient descent is used to minimize the mean square error between the estimate and the target value:

$$\begin{aligned} \theta_1 &\longleftarrow \theta_1 - \alpha\nabla_{\theta_1}\frac{1}{2}\left(Q_{\theta_1}(s_t, a_t) - Y(s_t, a_t)\right)^2, \\ \theta_2 &\longleftarrow \theta_2 - \alpha\nabla_{\theta_2}\frac{1}{2}\left(Q_{\theta_2}(s_t, a_t) - Y(s_t, a_t)\right)^2, \end{aligned} \tag{4}$$

where α represents the update step size. In the process of gradient updating, the target network parameters $\theta_{targ,1}$ and $\theta_{targ,2}$ are kept constant to ensure the stability of updating.

After the estimated network parameters are updated, the parameters of the target network are updated by soft update method. The formula is as follows:

$$\theta_{targ,1} \longleftarrow \tau\theta_1 + (1-\tau)\theta_{targ,1}, \tag{5}$$

$$\theta_{targ,2} \longleftarrow \tau\theta_2 + (1-\tau)\theta_{targ,2}, \tag{6}$$

where τ is the coefficient of soft update method. For the parameter ϕ of actor network, the gradient update direction is to maximize the distance between the current policy and π_η while maximizing the cumulative return. The distance between π_η and the current policy can be calculated by using the square of the maximum mean discrepancy (MMD).

Given samples $x_1, \ldots, x_n \sim P$ and $y_1, \ldots, y_m \sim G$, the square of the MMD can be estimated only from the sample of the distribution. Then, the square of MMD between distribution P and G can be written as

$$\mathrm{MMD}^2\left(\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_m\}\right) = \frac{1}{n^2}\sum_{i,i'} k(x_i, x_{i'}) - \frac{2}{nm}\sum_{i,j} k(x_i, y_j) + \frac{1}{m^2}\sum_{j,j'} k(y_j, y_{j'}), \tag{7}$$

where $k(\cdot,\cdot)$ is the kernel function. Here, Gaussian kernel is used in DEPRL, that is,

$$k(x_i, x_{i'}) = \exp\left(-\frac{\|x_i - x_{i'}\|^2}{2\sigma^2}\right), \quad \sigma > 0, \tag{8}$$

where σ is standard deviation. Record the square of MMD between policy π_η and policy π_ϕ as $D_{\mathrm{MMD}}(\pi_\eta, \pi_\phi)$, and the formula is as follows:

$$D_{\mathrm{MMD}}\left(\pi_\mu, \pi_\phi\right) = \mathrm{MMD}^2\left(\pi_\mu(\cdot|s), \pi_\phi(\cdot|s)\right)\ s \sim D, \tag{9}$$

where D is the experience pool.

To sum up, the objective function of actor network only considering the maximum cumulative return is

$$J_\pi(\phi) = E_{s\sim D, a\sim\pi_\phi(\times|s)}\left[Q_{\theta_1}(s,a)\right]. \tag{10}$$

When $D_{\mathrm{MMD}}(\pi_\eta, \pi_\phi)$ that satisfies the gradient update requirement is obtained, the objective function of the actor network can be written as

$$\begin{aligned} J_{\mathrm{MMD}}(\phi) = {} & \mathbb{E}_{s\sim D, a\sim\pi_\phi(\cdot|s)}\left[Q_{\theta_1}(s,a)\right] \\ & + \beta\mathbb{E}_{s\sim D}\left[\mathrm{MMD}^2\left(\pi_\mu(\cdot|s), \pi_\phi(\cdot|s)\right)\right], \end{aligned} \tag{11}$$

where $\beta > 0$ is the weighting factor. The number of actors that only consider cumulative returns is recorded as K_1, and then the number of actors that maximize $D_{\mathrm{MMD}}(\pi_\eta, \pi_\phi)$ at the same time is $K/2 - K_1$.

3.2. The Framework of DEPRL

3.2. The Framework of DEPRL. In CEM-RL method, the total number of individuals in the population is set to K. The mean μ and covariance matrix Σ of the policy parameter distribution are obtained by random initialization. According to the covariance matrix and the mean value, K parameters are extracted from the distribution as the parameters of actor network in the population. The actor network with half of the total number of individuals in the population is randomly selected for gradient update according to the value function output from critic network. The goal is to maximize the cumulative return of the actor network's corresponding policy. The critic network that guides actor network gradient updates throughout the process is the same; that is, half of the actors in the

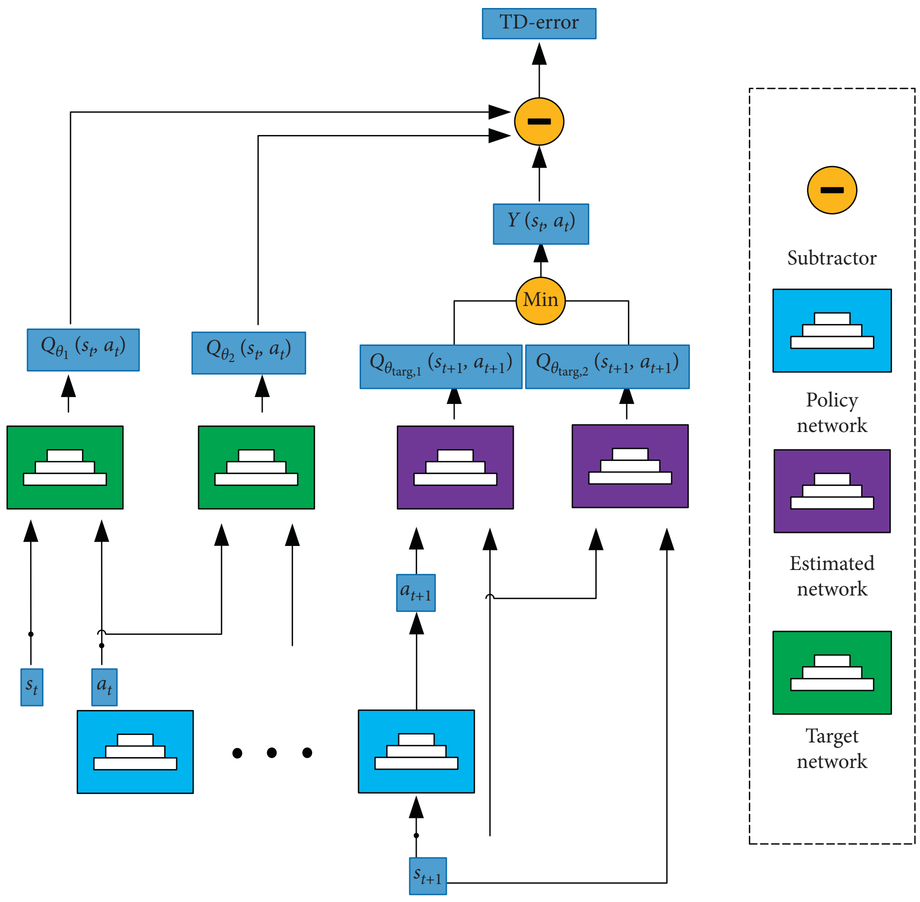


FIGURE 1: The update process of critic networks in DEPRL.

population use the same critic network to guide updates. In a population, the data generated by the interaction between the actor and the environment is stored in the experience pool and is used to train the critic network. By evaluating the cumulative returns of the policies corresponding to all actors in the population after gradient updating, the policies ranked in the top half of the cumulative returns are selected as the elite sample. The number of the elite sample K_e is usually set to $K_e = K/2$. Finally, according to the parameters of contemporary elite samples, μ_{new} and Σ_{new} of the new generation actor network parameter distribution are generated.

The framework of DEPRL algorithm is shown in Figure 2. Assume that the corresponding policy of Actor$_\mu$ composed of elite sample parameters is π_η. When the critic network guides the next generation policy update, it needs to maximize the MMD between a part of policies and π_η. By increasing the diversity of descendant policies, more space is explored, and the probability of the algorithm falling into the local optimal solution is reduced. When selecting the elite sample, not only the cumulative return of each policy should be considered, but also the MMD between each policy and π_η should be considered. In the population, the updated new policy is first sorted according to the cumulative return from high to low, and the policies with cumulative return ranked between 2 and $K/2$ greater than $\pi\mu$ cumulative return are taken out, and the MMD values between these policies and

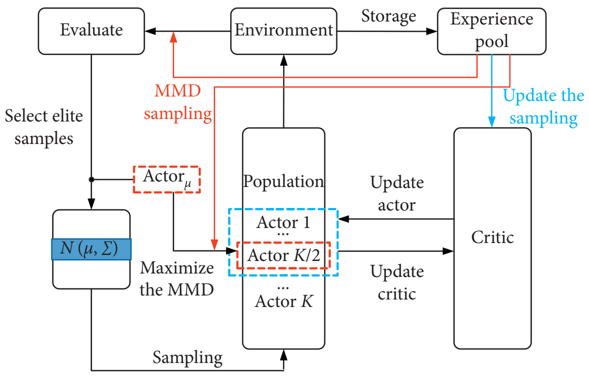


FIGURE 2: The framework of DEPRL.

π_η are calculated, and reorder the MMD value from largest to smallest. In the population, the updated new policy is first sorted according to the cumulative return from high to low. Then, the policies in which the cumulative return is between 2 and $K/2$ greater than the cumulative return of π_η are taken out. Finally, the MMD values between these policies and π_η are calculated. These policies are reordered in descending order of MMD value.

Use MMD as the standard to select policies that is quite different from $\pi\mu$ among contemporary policies, which helps transfer the diversity policy to the next generation distribution. The new generation policy generated by sampling in the new distribution is quite different from the old policy, which makes the trajectory of the new generation policy more diversified and can increase the exploration space. In order to reduce the amount of calculation when calculating the new distribution parameters, Σ is constrained to be a diagonal matrix. The update formulas of the new distribution parameters μ_{new} and Σ_{new} are as follows:

$$\mu_{\text{new}} = \sum_{i=1}^{K_e} \lambda_i z_i, \tag{12}$$

$$\Sigma_{\text{new}} = \sum_{i=1}^{K_e} \lambda_i \left(z_i - \mu_{\text{old}}\right)\left(z_i - \mu_{\text{old}}\right)^T + \varepsilon, \tag{13}$$

where λ_i represents the weight of the parameter corresponding to the i-th elite policy in the population, and ε is the Gaussian noise. λ_i can be defined as

$$\lambda_i = \frac{\left(\log\left(1+K_e\right)/i\right)}{\sum_{i=1}^{K_e}\left(\log\left(1+K_e\right)/i\right)}. \tag{14}$$

The above formula indicates that the higher the ranking of the parameters corresponding to the elite policy, the greater the value of a λ_i.

To sum up, the update process of DEPRL can be simply summarized as follows: (1) the parameter distribution of the initialization policy is N (μ_0, $\sum_0$); (2) K group policies are randomly selected corresponding to K group parameters from the distribution; (3) gradient updating is performed by randomly selecting $K/2$ policy; (4) the fitness of the corresponding policy under the K set of parameters is calculated; (5) the parameters corresponding to the current elite policy are used to calculate the parameter distribution (μ, $\sum$) of the next generation policy, as shown in equations (12) and (13); (6) whether the parameter distribution of the contemporary policy meets the requirements is determined; if so, stop updating; if not, repeat step (2).

The pseudocode of DEPRL algorithm is shown in Algorithm 1.

4. Results and Analysis

4.1. Experiment Settings. In this section, we use the MuJoCo test environment implemented in OpenAI Gym [31] to evaluate the performance of the proposed algorithm and comparison Algorithms. Gym is a basic platform for testing deep reinforcement learning algorithms provided by OpenAI. It provides a large number of simple interfaces for the training of the agent, greatly simplifies the interaction process between the agent and the environment, and facilitates related researchers to implement deep reinforcement learning algorithms and test the performance of deep reinforcement learning algorithms. Figure 3 shows the corresponding status screens of the four tasks in the MuJoCo test environment. Table 1 describes the state dimension and action dimension of the four tasks in the MuJoCo test environment, as well as specific task goals. According to the state dimension and action dimension information provided by MuJoCo, it is convenient to design the corresponding neural network for learning. The version of OpenAI Gym used in the experiment is 0.17.3, and the version of MuJoCo is 2.0.

Experiment settings are set up as follows:

(1) We chose to compare TD3, multiactor TD3, CEM, and CEM-TD3 to verify the superiority of the proposed DEPRL. The common superparameter settings of the five algorithms are the same as shown in Table 2, and the total numbers of population individuals and elite individuals of CEM-TD3 and DEPRL are the same, 10 and 5, respectively. When DEPRL calculates D_{MMD}, the data size M extracted from the experience pool is 600, the number of Gaussian kernel function $m = n = 5$, and the value of K_1 is 4. The weighting factor β in the objective function J_{MMD} is 0.2 in the Ant-v2 environment, and 0.1 in all other test environments.

(2) In order to make a fair comparison between different algorithms, we combined CEM and TD3 to form CEM-TD3 algorithm for experiment. And the network structure used by CEM to represent policies is consistent with that of DEPRL, CEM-TD3, multiactor TD3 and TD3. Multiactor TD3 is a variant of TD3. Compared with TD3, multiactor TD3 has multiple actors. The experience data generated by the interaction between multiple actors and the environment are sent to the experience pool together, and the critic remains unchanged. In the experiment, the number of actors in multiactor TD3 is set to 5, and the total number of gradient updates of critic and actor in multiactor TD3, CEM-TD3, and DEPRL is the same.

(3) We selected four environments HalfCheetah-v2, Hopper-v2, Walker2d-v2, and Ant-v2 for comparison, and the details of the test environment are shown in Table 1. The experimental results are shown in Figure 4, where the horizontal axis represents the number of time steps, and the vertical axis represents the cumulative return value of a round in the evaluation stage. During the training process, the performance of the current algorithm is evaluated every 1000 steps. Each algorithm was repeated with five different random seeds in different test environments. When drawing the reward curve, the sliding window size is set to 100. The curve part and shaded part in the figure represent the mean value and the standard deviation of the accumulated return value under multiple random seeds, respectively. We also present the mean and standard deviation of the cumulative return per turn in different MuJoCo tasks. The results can be found in Table 3.

4.2. Analysis of Experimental Results

(1) As can be seen from Figure 4, DEPRL performs best overall in the test environment and also performs best in the environment with higher state dimension

```
Input: the coefficient of soft update method $\tau$, sampling size of the experience pool $N$ and $M$, maximum number of time steps $T_{\max}$,
       discount factor $\gamma$, experience pool capacity $\mathbf{\Delta}_{\text{size}}$, population parameter $K$ and $K_1$
Output: actor network parameters $\phi^*$ corresponding to the optimal policy $\pi^*$
 (1) Initialize critic network parameters $\theta_1$, $\theta_2$, $\theta_{\text{targ},1}$, $\theta_{\text{targ},2}$ and actor network parameter distribution $(\mu_0, \sum_0)$
 (2) $T_{\text{total}} = 0$, $T_{\text{actor}} = 0$
 (3) WHILE $T_{\text{total}} < T_{\max}$:
 (4) Extract $K$ sets of parameters $para$ from the current distribution $(\mu, \sum)$
 (5) FOR $k = 1$ TO $K/2$:
 (6)   Initialize the actor according to the parameter $para[k]$
 (7)   FOR $t = 1$ TO $2 * T_{\text{actor}}/K$:
 (8)     Sampling $N$ samples from $\mathbf{\Delta}$ to minimize the objective function (3)
 (9)     Update $\theta_{\text{targ},1}$ and $\theta_{\text{targ},2}$ through equations (5) and (6)
(10) FOR $k = 1$ TO $K_1$:
(11) Initialize the actor according to the parameter $para$ $[k]$
(12) FOR $t = 1$ TO $T_{\text{actor}}$:
(13)     Sample $N$ samples from $\mathbf{\Delta}$ to maximize the objective function (11)
(14)     Replace the original parameter $para$ $[k]$ with the new actor parameter
(15) FOR $k = K_1 + 1$ TO $K/2$:
(16) Initialize the actor according to the parameter $para$ $[k]$
(17) FOR $t = 1$ TO $T_{\text{actor}}$:
(18)     Sample $N$ samples from $\mathbf{\Delta}$ to maximize the objective function (12)
(19)     Replace the original parameter $para$ $[k]$ with the new actor parameter
(20) $T_{\text{actor}} = 0$
(21) FOR $k = 1$ TO $K$:
(22) Initialize the actor according to the parameter $para[k]$
(23)     Interact with the environment to calculate the cumulative payoff $G$ and the total number of time steps used $T_{\text{episode}}$
(24)     Store data $(s, a, s', r)$ in the experience pool $\mathbf{\Delta}$
(25)     Sample $M$ samples from $\mathbf{\Delta}$ to calculate the $D_{\text{MMD}}$ between them and $\text{Actor}_\mu$
(26)     $T_{\text{actor}} = T_{\text{actor}} + T_{\text{episode}}$
(27)     $T_{\text{total}} = T_{\text{total}} + T_{\text{actor}}$
(28)     Select elite samples according to $G$ and $D_{\text{MMD}}$, and update the distribution according to equations (12) and (13)
(29) END WHILE
```

Algorithm 1: DEPRL.

and action dimension, such as Ant-v2 and Walker2d-v2. CEM performs worst overall and learns few effective policies in environments with higher state and action dimensions. Therefore, it can be shown that both the sample utilization and learning rate of CEM are significantly lower than those of other algorithms based on single-step update.

(2) In order to explore whether the improvement of DEPRL effect is due to the adoption of multiactor structure, we tested the influence of multiactor structure on the algorithm. Compared with the traditional actor-critic structure, the training data used by the critic in the multiactor structure is generated by the interaction between multiple actors and the environment. By comparing the reward curves of TD3, multiactor TD3, and DEPRL in Figure 4, it can be found that the reward curve of multiactor TD3 is only slightly higher than that of TD3 based on the traditional actor-critic structure. Therefore, it can be explained that the multiactor structure does not improve the algorithm much. In the Hopper-v2 training environment, multiactor TD3 began to oscillate when the cumulative return of the policy reached about 3200 and could not learn a better policy, while DEPRL with the same multiactor structure could get about 3600 cumulative returns. By comparing the reward curves among TD3, multiactor TD3, and DEPRL, it can be shown that the performance improvement of DEPRL does not simply depend on the multiactor structure.

(3) To explore the benefits of DEPRL in encouraging offspring diversity, we compared it with CEM-TD3, which only uses cumulative returns as a policy learning goal. CEM-TD3 also uses multiactor structure, and the total number of population individuals and the number of elite individuals is set the same as DEPRL. It can be seen from Figure 4 that the reward curve of DEPRL is significantly higher, and the reward curve of CEM-TD3 gradually levelled off in the second half due to the decline of exploration ability. Except for the Hopper-v2 test environment, DEPRL still maintained a relatively high growth trend in the second half of the reward curve.

(4) As can be seen from Table 3, the DPERL algorithm has the highest mean cumulative return of all the algorithms. The CEM algorithm performs the worst, which once again demonstrates that CEM, as a turn update algorithm with no experience replay,

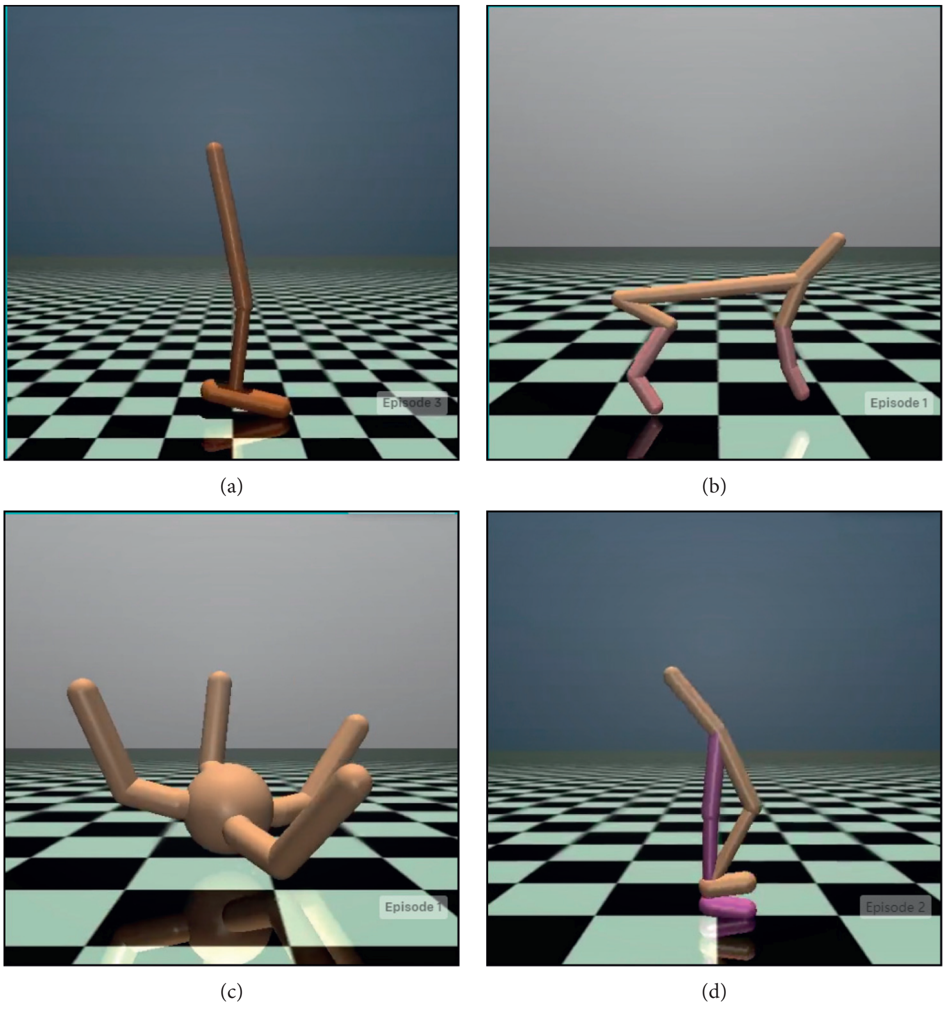


Figure 3: MuJoCo test environments. (a) Hopper-v2, (b) HalfCheetah-v2, (c) Ant-v2, and (d) Walker2d-v2.

Table 1: The test environment in the MuJoCo benchmark.

Environment	Action dimension/state dimension	Task goals
Hopper-v2	3/11	Make a two-dimensional one-legged robot hop forward as fast as possible
HalfCheetah-v2	6/17	Make the 2D cheetah robot run fast
Ant-v2	8/111	Make a four-legged creature walk forward as fast as possible
Walker2d-v2	6/17	Make a two-dimensional bipedal robot walk forward as fast as possible

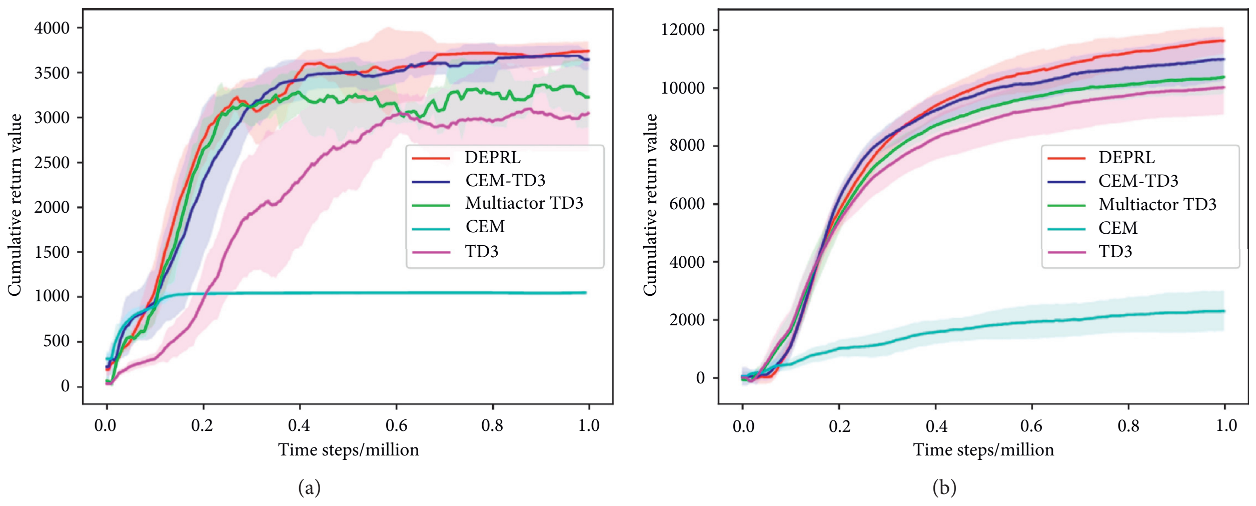
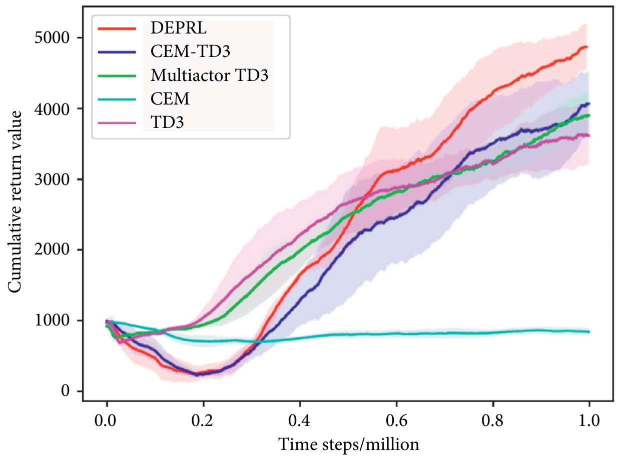
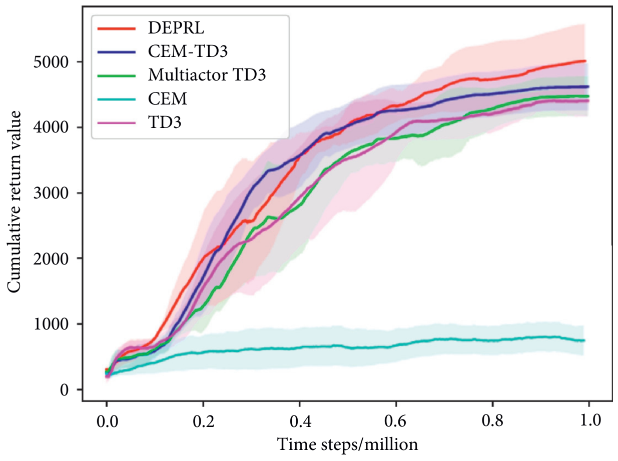


Figure 4: Continued.



(c)



(d)

Figure 4: Results of each algorithm in MuJoCo test environment. (a) Hopper-v2. (b) HalfCheetah-v2. (c) Ant-v2. (d) Walker2d-v2.

Table 2: Values of hyperparameter.

Hyperparameter	Values
Critic/actor learning rate	0.0003
Critic/actor hidden layer	2
Number of neurons	400/300
Critic activation	Relu
Actor activation	Tanh
Discount factor	0.99
Optimizer	Adam
Soft update coefficient	0.005
Experience pool capacity	10^6
Experience pool sample size	100
Gauss noise	Clip ((0, 0.2), −0.5, 0.5)

Table 3: The mean and standard deviation of the cumulative return per turn in different MuJoCo tasks.

Task	TD3	Multiactor TD3	CEM	CEM-TD3	DPERL
Hopper-v2	3025 ± 577	3241 ± 363	1054 ± 17	3652 ± 116	3732 ± 106
HalfCheetah-v2	10002 ± 930	10341 ± 578	2298 ± 690	10978 ± 758	11615 ± 464
Ant-v2	3618 ± 425	3881 ± 319	845 ± 52	4037 ± 466	4852 ± 317
Walker2d-v2	4399 ± 238	4470 ± 301	743 ± 225	4612 ± 357	5001 ± 562

could not learn effective strategies. Compared with TD3 and multiactor TD3 algorithms, DPERL and CEM-TD3 algorithms have higher average cumulative returns, which is due to the addition of evolutionary strategy into DPERL and CEM-TD3 algorithms. Compared with the CEM-TD3 algorithm, the DPERL algorithm achieves better results, because it increases exploration by encouraging the generation of diversity strategies in the offspring. In addition, in the Hopper-v2, HalfCheetah-v2, and Ant-v2 test environments, DPERL has smaller standard deviations than TD3, multiactor TD3, and CEM-TD3 algorithms, which indicates that DPERL algorithm has more stable results than the other three algorithms. To some extent, this also shows that DPERL algorithm can explore more effective strategies.

The above results clearly show that DEPRL improves the exploration ability of reinforcement learning agents and, to some extent, reduces the risk of policy updating falling into local optimum due to the disappearance of gradient.

5. Conclusions and Discussions

In this paper, we propose the DEPRL algorithm, which combines CEM and TD3 to measure the distance between different policies through MMD method. Some contemporary policies maximize the cumulative return while maximizing the distance between them and the previous generation policies and obtain policies with large differences to increase the scope of exploration. In the course of evolution, combining the cumulative return of a contemporary policy with the distance between the previous generation's policy as fitness helps the next generation's policy have more diversity based on a higher cumulative return. By combining TD3 with gradient updating and CEM without gradient updating, DEPRL can reduce the risk of policy updating falling into local optimal solution due to gradient disappearance by encouraging the generation of diversified policies in the offspring. By comparing DEPRL with CEM-RL, TD3, CEM, and multiactor TD3 in MuJoCo test environment, the experimental results show that DEPRL achieves more effect without increasing the number of update steps.

In DEPRL, we use an estimation of distribution algorithm to estimate the distribution of the elite samples and then select the elite samples that meet certain conditions to improve the diversification of the elite strategy. Except for estimation of distribution algorithms, some of the most representative computational intelligence algorithms can be used to reinforcement learning. Monarch butterfly optimization (MBO) [32] algorithm generates offspring by migration operator, which can be adjusted by the migration ratio of monarch butterflies. It is followed by tuning the positions for other butterflies by means of butterfly adjusting operator. In reinforcement learning, MBO can adjust the selection of elite samples in the global scope to avoid the loss of potential elite samples. In earthworm optimization algorithm (EWA) [33], the offspring are generated through Reproduction 1 and Reproduction 2 independently, and then, the weighted sum of all the generated offspring is used to get the final earthworm for next generation. Reproduction 1 generates only one offspring by itself that is also special kind of reproduction in nature. Reproduction 2 is to generate one or more than one offspring at one time. EWA can be used to replicate elite samples to ensure the high efficiency of elite strategies in reinforcement learning and speed-up learning. In elephant herding optimization (EHO) [34], the elephants in each clan are updated by its current position and matriarch through clan updating operator. It is followed by the implementation of the separating operator, which can enhance the population diversity at the later search phase. EHO is an appropriate way to increase the diversity of a population. Not only can it be used to eliminate bad reinforcement learning strategies, but it can also be used to add new strategies that did not exist before. Exploration is a vital part of reinforcement learning. Exploratory algorithms in computational intelligence algorithms can provide meaningful guidance for reinforcement learning. For example, slime mould algorithm (SMA) [35] uses adaptive weights to simulate the process of producing positive and negative feedback of the propagation wave of slime mould based on bio-oscillator to form the optimal path for connecting food with excellent exploratory ability and exploitation propensity. According to the moth's phototaxis and Levy flight characteristics, moth search (MS) [36] algorithm can do exploitation and exploration at the same time and ensures local search and global search. Harris Hawks Optimizer (HHO) [37] is a popular population-based nongradient optimization algorithm, which has many active time varying exploration and development stages. It has strong global searching ability.

We only analyzed the possibilities of the above computational intelligence algorithms in reinforcement learning applications, but these algorithms are not really used in reinforcement learning. Therefore, in the future work, we will devote ourselves to applying computational intelligence algorithms to strategy optimization, exploration enhancement, and acceleration of learning speed in reinforcement learning.

Data Availability

The data used to support the findings of this study are available from the first author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest in this work.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 61906198) and the Natural Science Foundation of Jiangsu Province (Grant no. BK20190622).

References

[1] R. Sutton and A. Barto, "Reinforcement learning: an introduction," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 285-286, 2005.

[2] H. Wang, Y. Gao, and X. G. Chen, "Transfer of reinforcement learning: the state of the art," *Acta Electronica Sinica*, vol. 36, no. S1, pp. 39–43, 2008.

[3] T. G. Dietterich, "Machine-learning research," *AI Magazine*, vol. 18, no. 4, p. 97, 1997.

[4] M. I. Jordan and T. M. Mitchell, "Machine learning: trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[5] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the International Conference on Machine Learning*, pp. 387–395, Beijing, China, June 2014.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, https://arxiv.org/abs/1707.06347.

[7] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *Computer Science*, vol. 3, pp. 1889–1897, 2015.

[8] C. Tessler, G. Tennenholtz, and S. Mannor, "Distributional policy optimization: an alternative approach for continuous control," 2019, https://arxiv.org/abs/1905.09855.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning," 2015, https://arxiv.org/abs/1509.02971.

[10] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, https://arxiv.org/abs/1802.09477.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, https://arxiv.org/abs/1801.01290.

[12] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[13] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” 2016, https://arxiv.org/abs/1509.06461.

[14] M. Fortunato, M. G. Azar, B. Piot et al., “Noisy networks for exploration,” 2017, https://arxiv.org/abs/1706.10295.

[15] O. Vinyals, I. Babuschkin, W. M. Czarnecki et al., “Grandmaster level in starcraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[16] C. Y. Liu, Y. Q. Tan, C. A. Liu et al., “Application of multi-agent reinforcement learning in robot soccer,” *Acta Electronica Sinica*, vol. 38, no. 8, pp. 1958–1962, 2010.

[17] J. D. Williams, K. A. Atui, and G. Zweig, “Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning,” 2017, https://arxiv.org/abs/1702.03274.

[18] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, and J. Schmidhuber, “Natural evolution policies,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 949–980, 2014.

[19] S. Tim, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution policies as a scalable alternative to reinforcement learning,” 2017, https://arxiv.org/abs/1703.03864.

[20] S. Khadka and K. Tumer, “Evolutionary reinforcement learning,” 2018, https://arxiv.org/abs/1805.07917.

[21] A. Pourchot and O. Sigaud, “CEM-RL: combining evolutionary and gradient-based methods for policy search,” 2018, https://arxiv.org/abs/1810.01222.

[22] W. X. Yun, “Research progress of genetic algorithm,” *Application Research of Computers*, vol. 4, pp. 1201–1206, 2012.

[23] L. Pedro and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Springer, Berlin, Germany, 2001.

[24] M. Hauschild and M. Pelikan, “An introduction and survey of estimation of distribution algorithms,” *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[25] G.-G. Wang, M. Lu, Y.-Q. Dong, and X.-J. Zhao, “Self-adaptive extreme learning machine,” *Neural Computing and Applications*, vol. 27, no. 2, pp. 291–303, 2016.

[26] J. Yi, J. Wang, G. W. Hauschild, and M. Pelikan, “Improved probabilistic neural networks with self-adaptive strategies for transformer fault diagnosis problem,” *Advances in Mechanical Engineering*, vol. 8, no. 1, pp. 1–13, 2016.

[27] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[28] G. G. Wang, L. H. Guo, H. Duan et al., “The mode and algorithm for the target threat assessment base on elman-adaboost storing predictor,” *Acta Electronica Sinica*, vol. 40, no. 5, pp. 901–906, 2012.

[29] G. G. Wang, L. H. Guo, and H. Duan, “Wavelet neural network using multiple wavelet functions in target threat assessment,” *The Scientific World Journal*, vol. 2013, Article ID 632437, 7 pages, 2013.

[30] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen, “Detection of malicious code variants based on deep learning,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.

[31] G. Brockman, V. Cheung, L. Pettersson et al., “Openai gym,” 2016, https://arxiv.org/abs/1606.01540.

[32] G. Wang, S. Deb, and Z. Cui, “Monarch butterfly optimization,” *Neural Computing and Applications*, vol. 31, no. 7, pp. 1995–2014, 2019.

[33] G. G. Wang, S. Deb, and L. D. S. Coelho, “Earthworm optimisation algorithm: a bio-inspired metaheuristic algorithm for global optimisation problems,” *International Journal of Bio-Inspired Computation*, vol. 12, no. 1, pp. 1–22, 2018.

[34] G. Wang, S. Deb, and L. Coelho, “Elephant herding optimization,” in *Proceedings of the 2015 3rd International Symposium on Computational And Business Intelligence (ISCBI)*, Bali, Indonesia, December 2015.

[35] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, “Slime mould algorithm: a new method for stochastic optimization,” *Future Generation Computer Systems*, vol. 111, no. 3, pp. 300–323, 2020.

[36] G.-G. Wang, “Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems,” *Memetic Computing*, vol. 10, no. 2, pp. 151–164, 2018.

[37] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, “Harris hawks optimization: algorithm and applications,” *Future Generation Computer Systems*, vol. 97, pp. 849–872, 2019.