Enhancing Rolling Horizon Evolution with Policy and Value Networks

Xin Tong
School of Information Science and Technology, USTC
Hefei, China
txt@mail.ustc.edu.cn

Weiming Liu
School of Information Science and Technology, USTC
Hefei, China
weiming@mail.ustc.edu.cn

Bin Li [*]
School of Information Science and Technology, USTC
Hefei, China
binli@ustc.edu.cn

***Abstract*—Rolling Horizon Evolutionary Algorithm (RHEA) is an online planning method for real-time game playing; its performance is closely related to the planning horizon and the search cost allowed. In this paper, we propose to learn a prior for RHEA in an offline manner by training a value network and a policy network. The value network is used to reduce the planning horizon by providing an estimation of future rewards, and the policy network is used to initialize the population, which helps to narrow down the search scope. The proposed algorithm, named prior-based RHEA (p-RHEA), trains policy and value networks by performing planning and learning iteratively. In the planning stage, the horizon-limited search is performed to improve the policies and collect training samples with the help of the learned networks. In the learning stage, the policy network and value network are trained with the collected samples to learn better prior knowledge. Experimental results on OpenAI MuJoCo tasks show that the performance of the proposed p-RHEA is significantly improved compared to that of RHEA.**

***Index Terms*—Rolling Horizon Evolutionary Algorithm, Reinforcement Learning, Covariance Matrix Adaptation Evolution Strategy, MuJoCo tasks**

I. Introduction

Games make an excellent domain for testing Artificial Intelligence techniques, due to their varying complexity and wide range of problems presented. In real-time game domains, Rolling Horizon Evolutionary Algorithm (RHEA) [1], [2] has been viewed as a suitable alternative to MCTS [3], [4], both of which can be performed in real time with little or no domain knowledge. Recent research in RHEA [5], [6] is advancing this method to produce better results than the vanilla version and get closer to the dominant performance of MCTS.

RHEA approach encodes a sequence of actions into an individual, then adopts Evolutionary Algorithms (EA) to optimize the action sequences directly. The individuals are evaluated by simulating actions ahead using a Forward Model (FM). After the population has been evolved for some predefined budget, the agent selects the first action of the best individual as the move to take in the real game. Within the same budget of FM calls, the performance of RHEA is closely related to its planning horizon H. A shorter planning horizon allows RHEA to iterate more generations, thus has more opportunities to find a better solution. But at the same time, it considers only the short-term rewards, which probably weaken the agent's performance as the agent may be too greedy to overlook the long-term rewards. A longer planning horizon allows RHEA to plan from a more global perspective, but it can only iterate a few generations and the search is less accurate.

To tackle the above problem, in this paper, an algorithm named prior-based RHEA (p-RHEA) is proposed to enhance RHEA in two ways. First, a value network is introduced to estimate the future rewards after H steps, in order to provide a long-term view with limited planning horizon. Second, a policy network is trained to initialize the population for RHEA, which can narrow down the search to high-probability actions and save the search budget. These two neural networks are trained by performing planning and learning in an iterative way. In each cycle, p-RHEA performs planning in simulated games with the prior knowledge provided by policy and value networks, while the samples generated in search are collected to train the neural networks to learn better prior knowledge. By iteratively performing planning and learning, p-RHEA continuously gets better samples and better neural networks.

The rest of this paper is organized as follows: Section II briefly introduces the existing methods for games. Section III introduces the background closely related to our work. Section IV shows the details of the proposed p-RHEA algorithm, and section V gives the experimental setup and results analysis. Finally, further discussions are presented in Section VI.

II. Related Works

A. Model-based planning methods

Model-based planning approaches do not need training but require a Forward Model to allow the agent to simulate before taking a move. MCTS [3], [4] is a typical model-based planning method. In each simulation iteration, it performs four steps: selection, expansion, evaluation and backup. When the iterations have finished, the action with the highest average reward or the maximum number of visits will be executed. Vanilla RHEA [1], [2] optimizes a series of fixed-length action sequences with corresponding cumulative rewards as their fitness, then selects the first action of the best individual as the move to take. Some enhancements have been introduced to the vanilla version in [5], including bandit-based mutation, statistical tree, shift buffer and rollout. Furthermore,

[*] Corresponding author.
Code and data are available at https://github.com/for-xintong/p-RHEA.

dynamic length rollout is proposed in [7] to tackle sparse rewards. And two different seeding technologies, One Step Look Ahead Seeding (1SLA-S) and Monte Carlo Tree Search Seeding (MCTS-S), are explored in [6]. The results suggest that both seeding variants offer a significant improvement in performance compared with random initialization.

B. Model-free learning methods

Model-free learning approaches need training but do not require a Forward Model. The agent always goes from a reset state to a terminal state, then starts over. Model-free reinforcement learning (RL) methods are typical representatives of this category. Deep Q-network (DQN) [8] is a value-based RL method; it aims to approximate the optimal action-value function to make decisions. In contrast to value-based methods, policy-based RL methods such as Asynchronous Advantage Actor-critic (A3C) [9] and Proximal Policy Optimization (PPO) [10] directly parameterize the policy and update the parameters by performing gradient ascent on the expected value of the total cumulative reward. There are also gradient-free methods to optimize the parameters of the policy network, such as CEM [11] and NES [12]. These population-based approaches often face the problem called ”curse of dimensionality”, because each individual encodes a deep neural network. ERL [13] and CEM-RL [14] combine gradient-free and gradient-based methods for policy search. They are hybrid algorithms that leverage the population of an EA to provide diversified samples to train an RL agent, and reinsert the RL agent into the EA population periodically to inject gradient information into the EA.

C. Combined methods

Combined approaches here refer to methods that combine planning and learning, which can take advantage of both model-based and model-free methods. They learn from the samples generated by model-based planning, and the knowledge gained can, in turn, be used for a better search. AlphaGo [15] and AlphaGo Zero [16] use a neural network to guide the search of MCTS in the game of Go. The neural network improves the strength of tree search, resulting in higher quality move selection. POLO [17] combines local trajectory optimization with global value function learning and explains how approximate value function can help reduce the planning horizon and allow for better policies beyond local solutions. Model-based RL methods can also be put into this category. To leverage planning in unknown environments, PlaNet [18] and SimPLe [19] present a model-based agent that learns a latent dynamics model from image observations and chooses actions by fast planning in latent space.

III. Background

A. Rolling Horizon Evolutionary Algorithm

Here we introduce the main idea of RHEA approach and the implementation details. In each state s, RHEA can be viewed as facing a trajectory optimization problem:

$$\hat{\pi}(s) = \underset{a_{0:H-1}|s_0=s}{\arg\max}\, \mathbb{E}\Bigg[\sum_{t=0}^{H-1}\gamma^t r(s_t,a_t) + \gamma^H r_f(s_H)\Bigg] \quad (1)$$

where $r(s_t,a_t)$ represents the reward obtained by performing action a_t under state s_t and $r_f(s_H)$ represents a terminal or final reward function which can be estimated by performing rollouts. $\gamma \in (0,1)$ is the discount factor. A smaller γ means greater weights are given to the more recent rewards. We encode a sequence of actions into an individual and then use a specific EA to optimize the policy in (1). Fitness values are calculated by executing the sequence of actions of the individual using the Forward Model, until all actions are executed or a terminal game state is reached. As the sequence is optimized, denoted by $a^*_{0:H-1}$, the first action a^*_0 is executed, and the procedure is repeated.

In the initialization stage of the next cycle, a shift buffer technology [5] is adopted: each action sequence in the current population is shifted one position to the left and a new random action is added to the right to form a new individual.

B. Policy evaluation and policy improvement

Here we introduce some basic concepts of policy iteration [20], which are necessary for our p-RHEA. We first introduce a policy $\pi(s)$, it inputs the current state s and outputs the action that should be taken under state s. Then we introduce a value function $V_\pi(s)$ that estimates how good the state s is under policy $\pi(s)$. It should be noted that a value function is defined with respect to a particular policy. The value of state s under policy $\pi(s)$ is the average discounted reward accumulated by following the policy from the state:

$$V_\pi(s) = \mathbb{E}\Bigg[\sum_{t=0}^{\infty}\gamma^t r(s_t,\pi(s_t))\,\Big|\, s_0 = s\Bigg] \quad (2)$$

Given the policy, we can update its value function in any state s, which is called policy evaluation:

$$V^{new}_\pi(s) = \mathbb{E}\Big[r(s,\pi(s)) + \gamma V_\pi(s^{'})\Big] \quad (3)$$

where $s^{'}$ is the state encountered after state s under policy $\pi(s)$. The purpose of computing the value function for a policy is to help find a better policy:

$$\pi^{new}(s) = \underset{a}{\arg\max}\, \mathbb{E}\Big[r(s,a) + \gamma V_\pi(s^{'})\Big] \quad (4)$$

This process of getting a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called policy improvement. Once a policy $\pi(s)$ has been improved using $V_\pi(s)$ to yield a better policy $\pi^{new}(s)$, we can then compute $V^{new}_\pi(s)$ and use it again to yield an even better policy. We can thus obtain a sequence of improved policies and value functions. This way of finding an optimal policy is called policy iteration.

IV. Method

As mentioned before, the proposed p-RHEA method has two neural networks to learn and store prior knowledge compared to RHEA approach. The policy network $p(a|s;\theta)$ takes the current state as input and outputs the probability distribution of the action that should be taken under the state. The value network $V_\pi(s;\theta_v)$ also takes the current state as input but outputs a scalar which estimates the expected future cumulative reward from the state. We use θ and θ_v to represent the parameters of the policy network and value network, respectively. These two neural networks are initialized to random weights, and we do not share parameters between policy and value networks. The proposed p-RHEA method consists of two stages: training (Section IV-B) and real play (Section IV-A).

A. Online planning with the learned prior knowledge

Given the policy network and value network, we can run p-RHEA online, similar to the RHEA process. The objective of online planning is shown as follow:

$$\pi(s) = \underset{a_{0:H-1}|s_0=s}{\arg\max}\ \mathbb{E}\left[\sum_{t=0}^{H-1}\gamma^t r(s_t,a_t) + \gamma^H V_\pi(s_H;\theta_v)\right] \tag{5}$$

where s is the current state and $a_{0:H-1}$ represents the action sequence p-RHEA is trying to optimize. After optimization, the first action a_0 will be performed at current state, then p-RHEA continues to the next cycle. The online planning process of p-RHEA is concluded below:

1) Initialize the population using the prior probability provided by the policy network. Specifically, the current state s_0 is first fed into the policy network to obtain a probability distribution, then an action a_0 is sampled from the distribution and executed to reach the next state s_1. After H steps, a sequence of actions is sampled, which can be encoded into an individual. This process is repeated for NP (population size) times to get a population.
2) Evolve the population with a specific EA. Individuals are evaluated by performing the sequence of actions in the simulated environment. The fitness of an individual is defined as $R = \sum_{t=0}^{H-1}\gamma^t r(s_t,a_t) + \gamma^H V_\pi(s_H;\theta_v)$. Future rewards after H steps are estimated by the value network.
3) When the evolution is complete, we record the best individual $a^*_{0:H}$ and its fitness R^*. Then we take a step $a^* = a^*_0$, and the game moves to the next state.
4) Repeat the above steps until a terminal state is encountered, which indicates the end of the game.

We use CMA-ES [21] for action sequence optimization, which utilizes a multivariate Gaussian distribution to model the correlation between variables and is proved to perform well on continuous optimization problems.

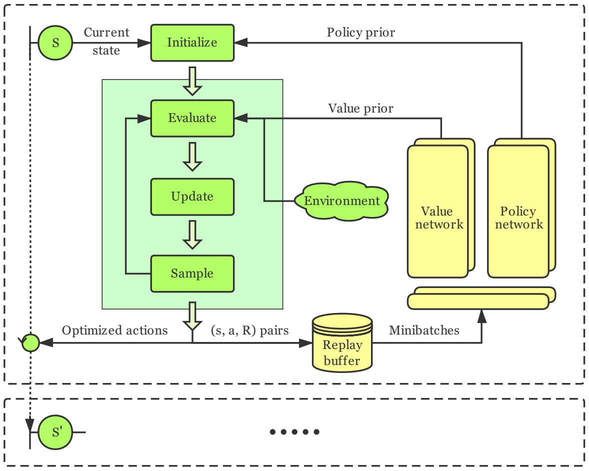

Fig. 1. High level training schematic of p-RHEA, highlighting the combination of planning (green) and learning (yellow)

B. Offline training to learn better prior knowledge

The training flow of the proposed p-RHEA approach is shown in Fig. 1, it performs planning and learning in an iterative way. In each simulation cycle, planning is first performed with the prior knowledge provided by policy and value networks. The sample pair (s, a^*, R^*) generated by the search is stored into a replay buffer D for training, with old experiences discarded if the buffer becomes full. The experience replay mechanism used here can remove correlations in the sample sequence [8]. From the perspective of RL, lookahead search can be similarly considered to be both a policy improvement operator and a policy evaluation operator: After the planning stage, we get an improved policy for state s and a new value estimate of state s under policy $\pi(s)$.

$$\pi^{new}(s) = a^*, \qquad V_\pi^{new}(s) = R^* \tag{6}$$

After every T steps of planning in the game and collecting experience, the policy network and value network are updated by learning from minibatches sampled uniformly from the replay buffer. In the learning stage, the parameters of the policy network θ are adjusted to maximize the similarity of the probability distribution $p(a|s;\theta)$ to the optimized action a^*. The parameters of the value network θ_v are adjusted to minimize the error between the new value estimate R^* and the old value estimate $V_\pi(s;\theta_v)$. Specifically, the neural networks are trained by RMSProp optimizer [22] on a loss function that sums over the maximum likelihood loss and mean squared error, respectively:

$$Loss = \mathbb{E}\left[-\log p(a^*|s;\theta) + \frac{1}{2}(R^* - V_\pi(s;\theta_v))^2\right] \tag{7}$$

The offline training of p-RHEA is a closed loop process. In the planning stage, the prior knowledge provided by the policy network and value network is used to achieve faster and more global search, while the sample pairs generated by search can in turn be used to train the networks in the learning stage to provide better prior knowledge. Through such iterations, we continually get better samples and better prior knowledge, and finally we can expect far better performance than in the beginning.

C. Summary of p-RHEA algorithm

The proposed p-RHEA algorithm has two distinct characteristics. (i) Policy prior: using a policy network to initialize the population. The initial population of p-RHEA is guided in a more promising region (may also get stuck in local optimum), thus can save many search costs compared to random initialization. Unlike seeding technologies such as 1SLA-S and MCTS-S, this method consumes mush less FM calls. But our method requires forward propagation of the policy network to add the prior knowledge into p-RHEA, which is not as fast as the shift buffer technology. (ii) Value prior: using a value network to estimate future rewards. With the help of the value network, p-RHEA can plan from a more global perspective and skip some local optimum while limiting the planning horizon. Compared to the rollout technology, our method does not require additional FM calls.

p-RHEA requires an additional training stage compared to RHEA, but once training is completed, p-RHEA is expected to achieve better performance with less search cost and shorter planning horizon in real play stage. Two tricks are used when training our p-RHEA. First, the value prior is added after a certain number of samples (more than or equal to V_{start}) have been collected, before which $V_\pi(s_H;\theta_v)$ can be considered equal to zero. This ensures the initial search will not be biased due to random initialization of the value network. A randomly initialized policy network does not introduce bias into the search and is effective from the beginning. Second, after the sequence $a^*_{0:H-1}$ is optimized, not only the first action but the first $T=\lfloor H/2 \rfloor$ actions will be executed. Correspondingly, the sample pairs $\{(s_t, a^*_t, R^*_t), t=0,1,..,T-1\}$ will be added to the replay buffer. One can refer to Algorithm 1 in the Appendix for more details. This change only occurs during the training stage, with the aim of speeding up the collection of samples and reducing the time required for training. The training performance of p-RHEA may be affected as a result, but good prior knowledge can still be learned for real play, as shown in the next section.

V. Experimental Studies and Results

In this section, we perform the evaluation and comparison of the proposed algorithm in several continuous control tasks. These tasks are simulated with the OpenAI MuJoCo physics engine [23] and aim to control the movement of various multi-joint robots, such as ant, swimmer, walker, etc. The states of the environment are the generalized positions and velocities of the simulated robots. At each step, the controller needs to control the simulated robot by setting the torques of the joints (i.e. each action is a real-valued vector), then the environment will simulate to the next state and return a reward. The controller has an access to the Forward Model, but the analytical expression of the reward is unknown.

A. Comparing p-RHEA with RHEA

RHEA with shift buffer is chosen as the baseline of this study. In order to compare p-RHEA with RHEA more thoroughly, two RHEA with different planning horizons are implemented. The former one has a planning horizon of 50 and a budget of 25000 FM calls, in order to reflect the extreme performance of RHEA. The latter one has a planning horizon of 20 and a budget of 4000 FM calls, in order to reflect the performance that is achievable in real time. CMA-ES is selected for trajectory optimization with a population size (NP) of 10. The means and standard deviations (calculated from 25 independent runs) of these two RHEA are listed in columns 2 and 3 of Table I, respectively.

The p-RHEA approach is implemented using CMA-ES as well, but only looks ahead 20 steps and is given a budget of 1000 FM calls. Considering that the forward propagation of the neural networks in p-RHEA takes some time, such a setting ensures that the proposed p-RHEA algorithm is faster in the real play stage than RHEA ($H=20$). It is worth noting that we distinguish the training budget from the playing budget in p-RHEA, because the training stage can be done offline and the game itself has some randomness at initialization. To represent the policy, we use a fully-connected multilayer perceptron (MLP) with two hidden layers of 128 units and ReLU nonlinearities to output the mean vector of the Gaussian distribution. The log-standard deviation is parameterized by a global vector independent of the state, as done in [24], [25]. The value network is also constructed with two hidden layers of 128 units and ReLU nonlinearities to output a scalar. For other parameters about the p-RHEA algorithm, please refer to Table II in the Appendix.

The training curves of p-RHEA, calculated from 5 independent runs, are shown in red in Fig. 2. As a comparison, we show the performance of RHEA (with shift buffer technology) in blue in Fig. 2. Note that the score curves of RHEA are horizontal because RHEA does not require learning. In order to be consistent with the setting of p-RHEA in the training stage, RHEA also optimizes a 20-step trajectory and then takes the first 10 steps. We can see that the performance of p-RHEA quickly surpasses that of RHEA through training, which means useful prior knowledge is being learned by the neural networks.

Then we test the performance of p-RHEA for real play. At this time, p-RHEA optimizes a 20-step trajectory and only takes the first action to execute. The means and standard deviations of 25 runs with the learned neural networks are listed in column 7 of Table I. To determine the statistical significance of the differences in performance, we performed a Wilcoxon rank-sum test [26] with Holm p-value adjustment and significance level of 0.05. The results that are significantly better than others are marked in bold.

When comparing the performance of RHEA with different search budgets, we can see that the results of looking ahead 50 steps are significantly better than those of looking ahead 20 steps, except on the Ant-v2 task. As mentioned before, a

TABLE I
PERFORMANCE OF P-RHEA AND COMPARISON ALGORITHMS ON MUJOCO TASKS

	RHEA+shift H=50	RHEA+shift H=20	policy only	p-RHEA (without value)	p-RHEA (without policy)	p-RHEA H=20
Ant-v2	2681 ± 299	**4891 ± 75**	593 ± 297	2593 ± 244	2011 ± 265	3229 ± 202
HalfCheetah-v2	**17586±1481**	6857 ± 379	103 ± 43	2169 ± 518	2916 ± 416	3162 ± 411
Hopper-v2	439 ± 82	285 ± 14	85 ± 53	339 ± 33	1466 ± 132	**2795 ± 727**
Humanoid-v2	2135 ± 1064	716 ± 113	152 ± 35	602 ± 96	3757 ± 1405	**4531 ± 878**
InvertedPendulum-v2	**1000 ± 0**	321 ± 257	695 ± 291	374 ± 223	172 ± 164	**1000 ± 0**
InvertedDoublePendulum-v2	3056 ± 1828	607 ± 118	5894 ± 4248	852 ± 41	375 ± 128	**9344 ± 3**
Swimmer-v2	52 ± 4	43 ± 2	130 ± 3	46 ± 2	71 ± 3	**138 ± 4**
Walker2d-v2	1089 ± 560	182 ± 39	308 ± 84	221 ± 65	2257 ± 1275	**3901 ± 639**

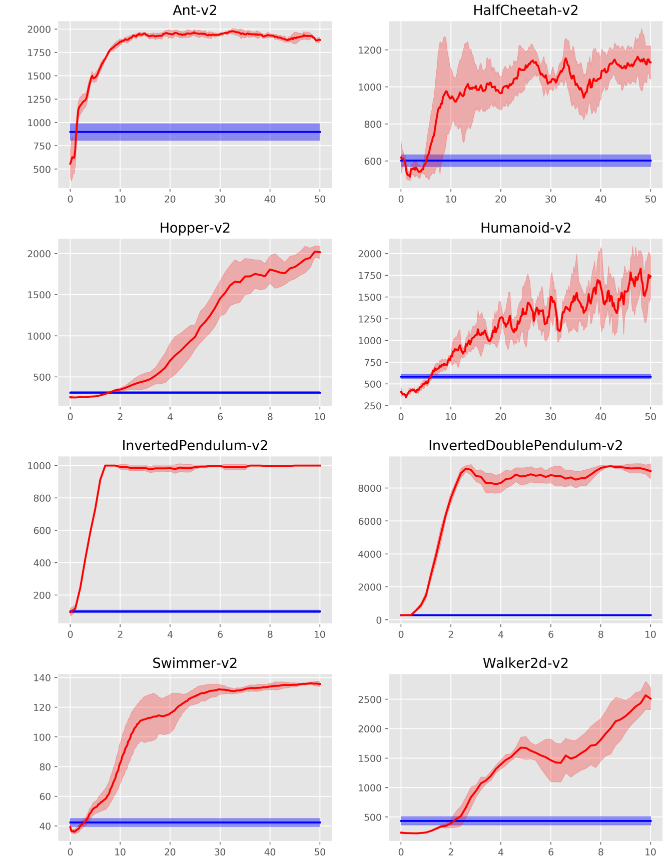

Fig. 2. Training curves of p-RHEA on the MuJoCo tasks (red), with the performance of RHEA as the control group (blue). The horizontal axis represents the number of FM calls (in millions) and the vertical axis represents the game score.

longer planning horizon allows RHEA to plan from a more global perspective and is expected for better performance. But for Ant-v2 task, it has eight joints to control. When looking ahead 50 steps, CMA-ES has to deal with a 400-dimensional problem which is quite difficult to optimize, and this may be the reason why the RHEA with longer planning horizon is worse than that with shorter planning horizon.

When comparing the performance of p-RHEA and RHEA, we can see that p-RHEA is generally better than RHEA approach, although under fewer FM calls. p-RHEA can benefit from the value prior and policy prior, which allow it to plan from a more global perspective and with less search cost. For example, without the global information provided by policy and value networks, the simulated robot controlled by RHEA will soon fall down due to seeking for high short-term rewards on the Hopper-v2 task and get trapped in local optimum due to avoiding big negative rewards on the Swimmer-v2 task. These two are typically tasks that involve deceptive rewards. In contrast, p-RHEA performs well on these two tasks by taking advantage of the prior knowledge learned, a video demonstration of how p-RHEA acts on the Hopper-v2 and Swimmer-v2 tasks is available here[1]. On the Ant-v2 and HalfCheetah-v2 tasks, however, p-RHEA behaves worse than RHEA. We checked carefully and found it is because p-RHEA looks ahead 20 steps and then take the first ten steps instead of the first one during training. This trick, as mentioned before, can reduce the time required for training. Taking ten steps per cycle performs well on other tasks, but on the Ant-v2 and HalfCheetah-v2 tasks, it makes the simulated robots sometimes fail to turn over. A video demonstration of how the number of steps taken can affect the performance on the HalfCheetah-v2 task is available here[2]. In conclusion, p-RHEA can benefit from the value prior to identify deceptive rewards and benefit from the policy prior to narrow down the search to high-probability actions, and finally shows a boost in performance over RHEA enhanced with shift buffer technology.

B. *Performance of p-RHEA with different components*

In order to study the role of the policy network and value network in p-RHEA separately, we evaluate three variants of p-RHEA. The first variant (V1) only uses the policy network for decision making. Specifically, the current game state is input into the policy network to obtain a Gaussian distribution of the action, and then the mean of the Gaussian distribution is taken as the next action to perform. This method does not require online planning and does not consume FM calls. The second variant (V2) removes the value network from p-RHEA, that is, set $V_\pi(s_H; \theta_v) = 0$. All other parameters are consistent with p-RHEA. The third variant (V3) removes the policy network from p-RHEA, which means that its population is randomly initialized. This method is actually a bit problematic because a value function is defined with respect to a particular policy. According to definition (2), $V_\pi(s; \theta_v)$ represents the average

[1] https://github.com/for-xintong/p-RHEA-video1
[2] https://github.com/for-xintong/p-RHEA-video2

discounted reward accumulated by following the given policy from the state s, so it cannot be used without the given policy or with a random policy. We list the results of the above three variants in the 4-6 columns of table I. The means and standard deviations are derived from 25 independent runs with the learned neural networks.

We can draw some useful conclusions. (i) The performance of V1, V2, and V3 is far worse than that of p-RHEA, which is natural because the policy network and value network are trained to work together. (ii) The performance of V2 is closer to that of RHEA ($H = 20$) compared to V1. For example, in the InvertedDoublePendulum-v2 and Swimmer-v2 tasks, although the policy networks are able to provide good population initialization, V2 will still choose the action that can maximize the short-term reward (thus result in a low game score) due to the lack of an estimate of future rewards in the optimization objective. (iii) V3 behaves relatively poorly when V1 performs well, such as on the InvertedDoublePendulum-v2 and Swimmer-v2 tasks. On these two tasks, the variance of the Gaussian distribution output by the policy network is small, so the policy itself is good enough to make decisions. But in the later stage of training, only a small part of the space is experienced and added to the replay buffer to train the value network. For many randomly initialized states, the value network is likely to give an unreliable value estimate and therefore V3 does not perform well. (iv) Conversely, V3 performs relatively well when V1 is poorly performing, such as on the Hopper-v2 and Humanoid-v2 tasks. On these two tasks, the variance of the Gaussian distribution output by the policy network is large, so the policy itself is not good enough to make a decision. But diverse samples are experienced and used for training the value network, so the value network learned is sufficient to evaluate a large part of the states in space. In other words, the learned value network can work well with a random policy.

C. *Visualizing the role of prior knowledge*

In order to visualize how the prior knowledge can help p-RHEA make better online planning, we draw the reward curves of the p-RHEA and RHEA ($H = 50$) on the Swimmer-v2 task, as shown in Fig. 3. The red lines represent the cumulative reward, and the blue lines represent the single step reward. Each single step reward is multiplied by 10 for better visualization. In addition, we also show the pose of the simulated robot at key time points.

An intuitive impression is that p-RHEA learned a fixed swimming pattern with the help of prior knowledge. The rewards obtained by p-RHEA are very periodic, but the rewards of RHEA seem to be irregular. For better analysis, we show the reward function of Swimmer-v2, which consists of two parts: a linear reward for forward progress v_x and a quadratic penalty on joint effort u.

$$r = v_x - 10^{-5}||u||^2 \tag{8}$$

The swimming cycle of p-RHEA can be divided into three stages. (i) Closing the legs, which will make the swimmer accelerate and result to a big positive reward. (ii) Floating, which uses inertia to advance and corresponds to a small positive reward. (iii) Opening the legs, which will produce a large joint effect. Due to the low speed, the penalty item plays a major role, and the environment returns a big negative reward. The actions of opening the legs which currently appear to be bad are chosen because the agent is informed by the value network that there will be greater positive rewards in the future. For example, when the cumulative reward is about 33, the swimmer controlled by p-RHEA jumps out of the reward trap with just one swimming cycle. In contrast, the swimmer controlled by RHEA does not open its legs significantly, so it is trapped in the local optimum, and the final swimming distance is very short.

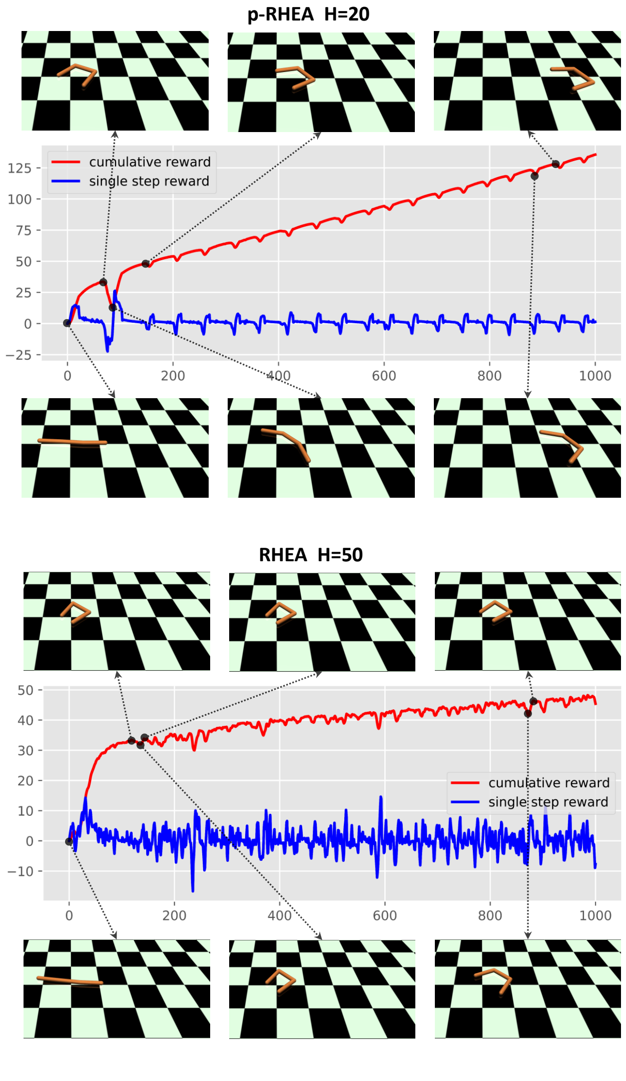

Fig. 3. Compare the rewards of p-RHEA and RHEA on Swimmer-v2

VI. Conclusions and Further Discussions

In this paper, we proposed a new method called p-RHEA for real-time game playing, which combines the strengths of planning and learning and shows clear advantages in continuous control tasks. In the training stage, p-RHEA iteratively executes planning and learning in simulated games. With such iterations, p-RHEA is constantly learning from its own experience and performing better and better. In the real play stage, p-RHEA uses the learned value prior to make more global planning while with a shorter planning horizon. In addition to this, p-RHEA uses the learned policy prior to narrow down the search to more promising region and can save considerable search costs.

In the experimental part, we first compared the RHEA and p-RHEA methods. Benefiting from the prior knowledge learned, p-RHEA can achieve better performance than RHEA with much less budget of FM calls, especially on tasks with deceptive rewards. Then the role of the policy network and value network in the p-RHEA algorithm is studied separately. We explained that the value network is only responsible for evaluating the states that policy network recommends to experience, so these two neural networks should work in coordination to show their power. Finally, we visually demonstrated that the prior knowledge learned can help p-RHEA plan from a more global perspective and jump out of local optimums on the Swimmer-v2 task.

Concerning with the future work, we are trying to test the performance of our p-RHEA algorithm in more complex environments like video games and two-player games. In addition, seeking for a better loss function seems to be a quite promising direction. Considering that CMA-ES can return not only the optimized action sequence but also the distribution of these actions, we can try cross-entropy loss for the update of the policy network and add additional constraints to make the policy update smoother.

Acknowledgment

The work is supported by the National Natural Science Foundation of China under grant NO.61836011 and NO.61473271.

References

[1] D. Perez-Liebana, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.

[2] R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, "Analysis of vanilla rolling horizon evolution parameters in general video game playing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 418–434.

[3] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo tree search: A new framework for game AI." in *AIIDE*, 2008.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Rolling horizon evolution enhancements in general video game playing," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 88–95.

[6] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, "Population seeding techniques for rolling horizon evolution in general video game playing," in *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 1956–1963.

[7] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Tackling sparse rewards in real-time games with statistical forward planning methods," in *AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] I. Szita and A. Lörincz, "Learning Tetris using the noisy cross-entropy method," *Neural computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[12] T. Rückstiess, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber, "Exploring parameter space in reinforcement learning," *Journal of Behavioral Robotics*, vol. 1, no. 1, pp. 14–24, 2010.

[13] S. Khadka and K. Tumer, "Evolutionary reinforcement learning," *arXiv preprint arXiv:1805.07917*, 2018.

[14] A. Pourchot and O. Sigaud, "CEM-RL: Combining evolutionary and gradient-based methods for policy search," *arXiv preprint arXiv:1810.01222*, 2018.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[17] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," *arXiv preprint arXiv:1811.01848*, 2018.

[18] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *arXiv preprint arXiv:1811.04551*, 2018.

[19] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski *et al.*, "Model-based reinforcement learning for Atari," *arXiv preprint arXiv:1903.00374*, 2019.

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

[21] N. Hansen, "The CMA evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[23] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

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

[26] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

Appendix

TABLE II
p-RHEA HYPERPARAMETERS USED ON MUJOCO TASKS

Hyperparameter	Value
Planning horizon (H)	20
Dimension (D)	H * action dim
Population size (NP)	10
Budget of FM calls (B)	1000
Steps taken per cycle (T)	1
Discount factor (γ)	0.99
Minibatch size	32
Replay buffer size (V)	20000
Replay start size (V_{start})	5000
Training times per cycle (NT)	50
RMSProp learning rate	3×10^{-3}
RMSProp decay factor	0.99
Gradient clipping	0.5

Algorithm 1 Pseudocode of p-RHEA for each cycle

```
Require: Current environment state $s$, Policy network $p(a|s;\theta)$, Value network $V_\pi(s;\theta_v)$,
    Budget of FM calls $B$, Planning horizon $H$, Population size $NP$,
    Discount factor $\gamma$, Training times $NT$, Replay buffer $D$, Replay start size $V_{start}$
Initialize the parameters of CMA-ES
while not exceeding the budget $B$ do
  if it is the first generation of p-RHEA then
    Initialize $NP$ $H$-steps action sequences using the policy network
    Encode each action sequence into an individual to form a population
  else
    Use the mean vector and covariance matrix of CMA-ES to generate a population
  end if
  for $i \leftarrow 1$ to $NP$ do
    Set environment state, i.e. $s_0 = s$
    Decode individual $i$ into an action sequence, $\{a_0, a_1, ..., a_{H-1}\}$
    for $t \leftarrow 0$ to $H-1$ do
      Interact with the environment: $r_t, s_{t+1} = Env.step(a_t|s_t)$
      if $s_{t+1}$ is a terminal state then
        break
      end if
    end for
    Length of legal action sequence: $L = t + 1$
    $R_L = \begin{cases} 0 & , s_L \text{ is a terminal state } || \text{ size of buffer } D < V_{start} \\ V_\pi(s_L;\theta), & \text{otherwise} \end{cases}$
    for $t \leftarrow L-1$ to $0$ do
      $R_t = r_t + \gamma R_{t+1}$
    end for
    Use $R_0$ as the fitness of individual $i$
  end for
  Select $\mu = \lfloor NP/2 \rfloor$ elite individuals to update the parameters of CMA-ES
end while
Record the optimized action sequence $\{a_0^*, a_1^*, ..., a_{L^*}^*\}$, state sequence $\{s_0, s_1, ..., s_{L^*}\}$and reward
sequence $\{R_0^*, R_1^*, ..., R_{L^*}^*\}$ found by CMA-ES
Steps taken per cycle $T = 1$
(for training, set $T = \max\{1, \lfloor L^*/2 \rfloor\}$ to speed up the collection of samples)
Set current environment state $s = s_T$
Put sample pairs $\{(s_t, a_t^*, R_t^*), t = 0, 1, .., T-1\}$ into the replay buffer $D$
for $j \leftarrow 1$ to $NT$ do
  if size of buffer $D \geq V_{start}$ then
    Randomly sample a minibatch from the replay buffer $D$
    Train the policy network and value network based on the loss function (7)
  end if
end for
return The current environment state $s$
```