Policy Return: A New Method for Reducing the Number of Experimental Trials in Deep Reinforcement Learning

FENG LIU\textsuperscript{1}, SHULING DAI\textsuperscript{1,2}, AND YONGJIA ZHAO\textsuperscript{1}

\textsuperscript{1}State Key Laboratory of VR Technology & Systems, Beihang University (BUAA), Beijing 100191, China
\textsuperscript{2}Jiangxi Research Institute, Beihang University (BUAA), Beijing 100191, China

Corresponding author: Feng Liu (by1503101@buaa.edu.cn)

This work was supported by the Major Project of the New Generation of Artificial Intelligence, China, under Grant 2018AAA0102900.

\section*{ABSTRACT}
Using the same algorithm and hyperparameter configurations, deep reinforcement learning (DRL) will derive drastically different results from multiple experimental trials, and most of these results are unsatisfactory. Because of the instability of the results, researchers have to perform many trials to confirm an algorithm or a set of hyperparameters in DRL. In this article, we present the policy return method, which is a new design for reducing the number of trials when training a DRL model. This method allows the learned policy to return to a previous state when it becomes divergent or stagnant at any stage of training. When returning, a certain percentage of stochastic data is added to the weights of the neural networks to prevent a repeated decline. Extensive experiments on challenging tasks and various target scores demonstrate that the policy return method can bring about a 10\% to 40\% reduction in the required number of trials compared with that of the corresponding original algorithm, and a 10\% to 30\% reduction compared with the state-of-the-art algorithms.

\section*{INDEX TERMS}
Deep reinforcement learning, policy return method, fewer trials, stochastic data.

\section*{I. INTRODUCTION}
Deep reinforcement learning is a combination of deep learning and reinforcement learning, thus enabling it to possess both the perception ability of deep learning and the policy-making ability of reinforcement learning [1], [2]. In recent years, models based on deep reinforcement learning have achieved excellent performances across various domains, for instance, beating the top human players in Go matches [3], [4], controlling the operation of complex machinery [5]–[8], allocating network resources [9], improving wireless communication technologies [10], [11], etc.

However, while deep reinforcement learning has made remarkable achievements, it has long been faced with the issue of having to undergo many trials due to unstable results [12], [13]. With appropriate multilayer perceptron function approximators and tuned hyperparameters, supervised learning can always acquire a satisfactory result. If the training process is performed multiple times, the learning curves acquire similar distributions. However, these phenomena do not occur with deep reinforcement learning. Most often, when using the same algorithm and hyperparameters, the results of multiple training processes differ widely in deep reinforcement learning. Even a DRL model that produces an excellent policy could, in all probability, result in a nonconvergent learning.

When training a DRL model, many researchers may encounter the dilemma that when using the same algorithm and hyperparameters, in some trials, the learning curves cannot converge after a prolonged training process or suddenly diverge from a previous convergence, but in other trials, they converge well. As a result, researchers have to perform many trials for a particular configuration and then choose the relatively best policy from the results.

There are many reasons for the instability of DRL. Strong correlations among state sequences easily lead to the divergence, and the data distribution that changes as the algorithm learns new behaviors also brings about troubles for DRL [12]. Experience replay and distributed training are used to reduce the adverse impacts caused by highly correlated samples and
changes in the data distribution [14]–[17]. However, according to our experience, a DRL algorithm using these methods may still have a high chance of diverging during the training process, and we highlight this in experiment 1.

For DRL, researchers often use random seeds to generate stochastic states at the beginning of an episode or add randomness to the action selection procedure for further exploration [18]. At the same time, a DRL model is often initialized by random weights within a certain range or obeying a given distribution. Environmental stochasticity and stochasticity in the learning process are crucial factors that lead to various results [12].

In recent years, researchers have made many efforts to make DRL competent for highly challenging tasks by optimizing algorithms [19]–[22], but few of them have paid attention to the large number of trials caused by unstable results. This article seeks to reduce the amount of required trials by introducing a method that allows the learned policy to return to a previous better state to increase the chance of convergence; we call this the policy return method. When returning, a certain percentage of stochastic weight is added to the DRL model to improve its exploration ability and avoid a repetitive decline. We conducted experiments on challenging tasks, and the results indicated that our proposed method can effectively reduce the number of trials when training a DRL model.

The remainder of the article is organized as follows. Section II introduces the technical background of this article. In Section III, we detail the policy return method. In Section IV, we describe detailed experimental settings, the results and discussions. Finally, we draw a conclusion from our research in Section V.

II. TECHNICAL BACKGROUND

In this section, we introduce reinforcement learning, deep reinforcement learning and some academic concepts that enable the comprehension of subsequent research procedures. A few DRL algorithms that are used in our experiments are also introduced.

A. RL AND DRL

Reinforcement learning (RL) is an area of machine learning that learns policy knowledge by exploring the environment [23]. RL follows the discounted Markov decision process \((S, A, \gamma, P, r)\) [24]. At step \(t\) during training, the algorithm chooses an action \(a_t \in A\) according to its policy \(\pi_t(a|s_t)\) based on the current state \(s_t \in A\) and then the environment offers a reward \(r(s_t, a_t)\) and the next state \(s_{t+1}\) according to the transition probability \(P(s_{t+1}|s_t, a_t)\). The discounted cumulative reward is defined as

\[
G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
\]

and the action value function \(Q(s, a)\) is defined as

\[
Q(s, a) = \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1}|s_t = s, A_t = a]
\]

Traditional RL algorithms often use a Q-table to record the Q-value that denotes the learned policy and is updated obeying the Bellman equation [25]. Q-learning [26], as a classic RL algorithm, is updated by

\[
Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]
\]

where

\[
V_n = \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]
\]

In 2013, DeepMind utilized a multilayer neural network to take place of the Q-table and they called it Q-network [1]. It is a classical work in the field of DRL. By differentiating the loss function with respect to the weights \(\theta\), the gradient of the Q-network is defined as

\[
\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s_t \sim \rho, s' \sim r}[Q(s_t, a_t|\theta_i^Q) - y_t] \nabla_{\theta_i} \mathbb{E}_{a_t \sim \mu_a(s|\theta_i^\mu)}[Q(s_t, a_t|\theta_i^Q)]
\]

With deep neural networks, reinforcement learning can handle diversified state data and even pixelated image information, and the networks also bring RL the capacity to handle complex continuous tasks.

B. DEEP DETERMINISTIC POLICY GRADIENT

Deep deterministic policy gradient (DDPG) [15] is a DRL algorithm that uses the actor-critic structure [27], and it generates deterministic actions rather than a policy distribution. DDPG algorithm has 4 networks: an actor \(\mu(s|\theta_i^\mu)\), a critic \(Q(s, a|\theta_i^Q)\), a target actor \(\mu'(s|\theta_i^\mu)\) and a target critic \(Q'(s, a|\theta_i'^Q)\). The weights of these target networks are updated by slowly tracking the learned networks: \(\theta' \leftarrow \tau\theta + (1-\tau)\theta'\), with \(\tau \ll 1\).

The critic \(Q(s, a|\theta_i^Q)\) is optimized by minimizing the loss:

\[
L_i(\theta_i^Q) = \mathbb{E}_{s_t \sim \rho, a_t \sim \beta, r_t \sim r}[Q(s_t, a_t|\theta_i^Q) - y_t]^2
\]

where

\[
y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta_i'^Q))
\]

The actor policy \(\mu(s|\theta_i^\mu)\) is updated by the sampled policy gradient:

\[
\nabla_{\theta_i} J \approx \frac{1}{N} \sum_{i} \nabla_{\theta_i} Q(s, a|\theta_i^Q)|_{s_0=a_0=\mu(s_0)} \nabla_{\theta_i} \mathbb{E}_{s_t=a_0=\mu(s_t)}[Q(s, a|\theta_i^Q)]
\]

DDPG is an outstanding DRL algorithm. In the experiment 3, we integrate policy return method into DDPG and perform testing on a challenging task to verify whether our proposed method works.

C. PROXIMAL POLICY OPTIMIZATION ALGORITHMS

Proximal policy optimization algorithms (PPO) [28] are policy gradient methods for reinforcement learning that optimize a “surrogate” objective function using stochastic gradient ascent. Similar to TRPO [29], PPO can choose an adaptive step size to update its policy under the premise of learning stability, and it can be easier to implement than TRPO. The newest PPO algorithms use a clipping method to replace the
Kullback-Leibler divergence that is used to restrict the step size of the update [28], and the loss function for updating the policy in PPO is the following:

$$L^{CLIP}(\theta) = \hat{E}_t \left[ \min(\hat{r}_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t) \right]$$

(9)

Here, $r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_t(a_t|s_t)}{\pi_{old}(a_t|s_t)}$, so $r(\theta_{old}) = 1$, and $\hat{A}_t$ is the advantage function, which is defined as

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{T-t} r_{T-1} + \gamma^{T-t} V(s_T)$$

(10)

where $V(s_t)$ specifies the value of the action that is provided by the critic network.

PPO is very good at continuous tasks and has become the main DRL algorithm of OPENAI. In experiment 2, we compare PPO on its own with PPO integrated into the policy return method on a continuous control task.

D. DISTRIBUTED PPO

Distributed PPO (DPPO) [17] is a modified PPO algorithm that uses multiple agents and multiple threads to speed up the training and reduce the correlations among samples. As Fig.1 shows, distributed PPO has some worker agents that learn in different environments and transmit learned policy gradients to the master agent. Every environment runs in an individual thread, and the weights of workers are updated in accordance with those of the master at regular intervals.

FIGURE 1. The work forms of multiple agents in distributed PPO.

In this study, we perform a few tests on distributed PPO in experiment 1 to observe the various results obtained based on a certain DRL algorithm and hyperparameter configurations.

III. POLICY RETURN METHOD

The policy return method makes copies of the networks in the original DRL algorithms, so it has two sets of networks with one called the optimal network and the other called the exploratory network. The optimal network is used to record the current optimal policy, and it is also the final policy for learning. The exploratory network is used to learn in the environment and transmit a better policy $\pi_{\theta e}$ to the optimal network. If the performance of the exploratory network deteriorates or cannot surpass that of the optimal network after a period of training, we make the exploratory network return to the policy of the optimal network by copying weights $\theta^o$ to $\theta^e$ and adding a certain percentage of the stochastic data $\sigma$ to the exploratory network. Fig.2 schematically shows the policy transmission between the two networks.

A. RECORDING THE OPTIMAL POLICY

In DRL, any tentative action reaps a reward from the environment, and researchers often use the episode-reward (EPR), which is the sum of the rewards for all steps in an episode, to represent the performance of a learning model. The EPR is calculated by

$$EPR = \sum_{s=1}^{M} r_s$$

(11)

where $M$ is the maximum step limit of an episode and $s$ is the index of the learning steps.

At time $t$, let $A$ denote the average of the latest $KEPRs$ of the exploratory network; $A$ is defined as

$$A = \frac{1}{k} \sum_{i=t-k+1}^{t} EPR_i$$

(12)

During training, when the $KTH$ episode is completed, the first $A$ is calculated. Let $A_{max}$ denote the current maximum $A$. Update $A$ every time an episode is completed, and if $A_{max} < A$, set $A_{max} = A$; then, copy the weights of the exploratory network to the optimal network ($\theta^o \leftarrow \theta^e$), so that the optimal network always records the current optimal policy.

B. POLICY RETURN

In this research, if the rewards of the exploratory network could not surpass those of the optimal network after $C$ episodes, the exploratory network will be forced to return to the optimal network that has the current optimal policy. The judgment for return is as follows:

$$\begin{align*}
A_{max} &> A_{t-C+c+1} \\
A_{max} &> A_{t-C+2} \\
& \vdots \\
A_{max} &> A_t
\end{align*}$$

(13)
Fig. 3 shows the process of policy return. After the decline lasted for $C$ episodes, at the pink point, the policy returned to the latest recorded point and continued to learn. The policy may not ameliorate immediately after the return, and here we only describe the probable effect of policy return simply. The actual experimental curves for policy return are shown in Fig. 6 and Fig. 8 in Sec.IV.

To avoid repeating a previous decline and improve the exploration ability of the model, we also add a certain percentage of the stochastic data $\sigma$ to the exploratory network. A combination of stochastic data that has the same mode with the weights $\theta_e$ is easy to acquire by reinitializing the exploratory network, so $\sigma = \theta_e^{\text{init}}$, and the return method is defined as

$$\theta^e \leftarrow \varphi \theta^o + (1 - \varphi)\theta_e^{\text{init}}$$

(14)

Here, $\varphi$ is the return ratio factor that controls the percentage of stochastic data, where $\varphi \ll 1$. Algorithm Box 1 shows the core policy return method in pseudocode.

### IV. EXPERIMENTS

#### A. EXPERIMENT 1: OBSERVING THE DRL RESULTS FROM TRIALS WITH THE SAME ALGORITHM AND HYPERPARAMETERS

To observe the difference between the results from multiple DRL trials that use a certain algorithm and hyperparameters, we performed a number of trials for the distributed PPO algorithm on the Pendulum-v0 task.

Environment: Pendulum-v0 is a classic continuous control task from OPENAI-GYM. The goal is to make a pole stand up by applying a torque, and Fig. 4 shows the motion sequences of a successful operation for Pendulum-v0. More information about Pendulum-v0 can be viewed at http://gym.openai.com.

Experimental Details: The DPPO algorithm that was used in this experiment is introduced in the Sec.II-D. We used 3 worker-agents running in different threads to learn in the environments. To train the weights of the neural networks, we used the Adam [30] optimization method and the learning rates $10^{-4}$ and $20^{-4}$ for the actor and critic, respectively. The actor network contained 2 hidden layers with 64*32 units, and the critic network contained one hidden layer with 128 units. Tab.1 shows more details about this experiment.

| Hyperparameter | Value     |
|----------------|-----------|
| structure (actor) | 64, 32 (relu, relu) |
| structure (critic) | 128 (relu) |
| learning rate (actor) | $10^{-4}$ |
| learning rate (critic) | $20^{-4}$ |
| worker-agents | 3 |
| discount factor $\gamma$ | 0.99 |
| clipping width $c$ | 0.2 |
| batch size | 64 |
| update steps | 10 |

**Algorithm 1 Policy Return**

1. Randomly initialize the optimal network and the exploratory network with weights $\theta^o$ and $\theta^e$.
2. Initialize $A$ (the average EPR of the latest $K$ episodes).
3. Initialize $A_{\text{max}}$.
4. Initialize the size of reward buffer $L_B$.
5. for episode $= 1$ to $N$ do
6. Initialize reward buffer $B$.
7. Initialize EPR.
8. for step $= 1$ to $M$ do
9. Run policy $\pi_{\theta^o}$, collecting $\{s, a, r\}$.
10. Update the exploratory network using the certain DRL algorithm.
11. Update $EPR$: $EPR = EPR + r$.
12. end for
13. Estimate $A$: $A = \frac{1}{k} \sum_{t=1}^{T} EPR_t$.
14. Store $A$ in $B$.
15. if $A > A_{\text{max}}$ then
16. Set $A_{\text{max}} = A$.
17. Update the optimal network: $\theta^o \leftarrow \theta^e$.
18. end if
19. if $A \leq A_{\text{max}}$ lasted $C$ episodes, see Eq(13) then
20. Getting weights $\theta_e^{\text{init}}$ by randomly initializing the exploratory network.
21. Return the exploratory network: $\theta^e \leftarrow \varphi \theta^o + (1 - \varphi)\theta_e^{\text{init}}$.
22. end if
23. end for
Results and Discussion: From Fig.5, we can see clearly that the same configurations produced very different results. Only half of the results (b, c) were convergent. (a) was divergent throughout the training process. (d) suddenly became divergent after achieving convergence, and the divergence was maintained until the end of training. For this experiment, we may choose policy (b) or (c) as the final result. However, if we only conduct one trial and obtain a terrible result such as that of (a) or (d), we will be very confused about whether to try again or abandon the current algorithms. A cruel reality is that researchers have to perform many trials when they are tuning a DRL algorithm. Therefore, it is significant to study how to verify a DRL algorithm or a batch of hyperparameters with fewer trials.

B. EXPERIMENT 2: COMPARISON OF PPO WITH PR-PPO ON PENDULUM-v0

In this study, we performed 50 trials for the original PPO and PPO with the policy return method (PR-PPO) on the Pendulum-v0 task, and each trial had 2000 learning episodes. The PPO algorithms used in this experiment are introduced in Sec.II-C.

Environment: We used the Pendulum-v0 environment to test our algorithm in this experiment and this environment was introduced in experiment 1.

Experimental Details: In this experiment, the trials of PPO and the trials of PR-PPO used the same configurations except for the exclusive hyperparameters ($A_{\text{max}}$, $L_B$, $C$, $\varphi$) of PR-PPO; therefore, they could prominently demonstrate the influence of the policy return method on DRL. The detailed configurations for this experiment are shown in Tab.2.

Results and Discussion: Fig.6 shows two learning curves from the PR-PPO trials. In panel (a), we can note that, in most cases, the returns at the position of pink dots can make the learning reach an improved state, and with the returns, the score of the optimal network becomes increasingly higher. In panel (b), in the first half of the learning process, the situation is difficult, but after a few policy returns, the learning
FIGURE 6. Learning curves obtained by PPO with the policy return method on the Pendulum-v0 task: panel (a) shows the capacity of policy return method to help the learning to obtain higher scores, and panel (b) mainly shows the process that the returns save the learning from a poor situation.

TABLE 2. Detailed Configurations for PPO and PR-PPO on the Pendulum-v0 Task.

| Hyperparameter       | PPO          | PR-PPO       |
|----------------------|--------------|--------------|
| structure (actor)    | 64, 32 (relu, relu) | 64, 32 (relu, relu) |
| structure (critic)   | 128 (relu)   | 128 (relu)   |
| learning rate (actor)| $10^{-4}$   | $20^{-4}$    |
| learning rate (critic)| $10^{-4}$ | $20^{-4}$    |
| optimization method  | Adam         | Adam         |
| discount factor $\gamma$ | 0.99        | 0.99         |
| clipping width $\epsilon$ | 0.2          | 0.2          |
| batch size           | 64           | 64           |
| update steps         | 10           | 10           |
| $K$                  | 100          | 100          |
| $N$                  | 2000         | 2000         |
| $M$                  | 300          | 300          |
| initial $A_{\text{max}}$ | -            | -2300        |
| $L_B$                | -            | 100          |
| $C$                  | -            | 350          |
| $\rho$               | -            | 0.97         |

stops diverging and finally gets back on track. Fig.7 shows a comparison between the average learning curve of the 50 PPO trials and the average learning curve of the 50 PR-PPO trials, and we can find that PR-PPO can not only make the DRL more stable but also obtain higher scores than those of PPO in the end.

Tab.3 lists the required number of trials ($N_r$) for various target scores. $N_r$ is estimated by $N_r = 50/w$, where $w$ is the number of trials whose best 100-episode performance surpasses the target score. Tab.3 indicates that the stable learning brought about by the policy return method leads to fewer required trials at all target score levels, and with the target score continuously increasing, this help becomes more and more significant.

C. EXPERIMENT 3: COMPARISON OF DDPG WITH PR-DDPG ON BipedalWalker-v2

To verify the general suitability of the policy return method with regard to DRL algorithms, we applied it to DDPG, which is a popular off-policy DRL method. Similar to experiment 2, we performed multiple trials for the original DDPG and DDPG with policy return (PR-DDPG) on a highly complicated task—BipedalWalker-v2.

Environment: BipedalWalker-v2 is a bipedal walking control task and is also from OPENAI-GYM. The goal of this

FIGURE 7. Comparison between the performances of PPO and PR-PPO on the Pendulum-v0 task.

TABLE 3. Statistical Results of Comparative Verification Experiments for PPO and PR-PPO on the Pendulum-v0 Task.

| Target score | Required number of trials on PPO | Required number of trials on PR-PPO |
|--------------|---------------------------------|-----------------------------------|
| -230         | 1.38                            | 1.21                              |
| -220         | 1.47                            | 1.42                              |
| -210         | 2.27                            | 1.78                              |
| -200         | 2.38                            | 1.85                              |
| -190         | 3.57                            | 2.00                              |
| -180         | 3.85                            | 2.17                              |
| -170         | 4.17                            | 2.50                              |
Figure 8. Learning curves from DDPG with the policy return method on the BipedalWalker-v2 task: panel (a) shows how policy return helps the learning obtain higher scores, and panel (b) shows that the returns can prevent the learning from falling into the divergence.

Table 4. Detailed Configurations for DDPG and PR-DDPG on the BipedalWalker-v2 Task.

| Hyperparameter       | DDPG         | PR-DDPG       |
|----------------------|--------------|---------------|
| structure (actor)    | 500, 200     | 500, 200      |
| structure (critic)   | 700 (relu)   | 700 (relu)    |
| learning rate (actor)| 5^-4        | 5^-4          |
| learning rate (critic)| Adam       | Adam          |
| optimization method  | Adam         | Adam          |
| batch size           | 32           | 32            |
| K                    | 50           | 50            |
| N                    | 2000         | 2000          |
| M                    | Till Done    | Till Done     |
| initial $A_{max}$    | -            | -100          |
| $G_{dir}$            | 50           | 36            |
| $C$                  | -            | 0.98          |

Experimental Details: For DDPG method, the actor network has 2 hidden layers with 500*200 units, and the critic network has one hidden layer with 700 units. In order to improve the sampling efficiency a prioritized experience replay method [31] was used in choosing transitions. Tab.4 shows the details about this experiment.

Results and Discussion: From the results demonstrated by Fig.8, the policy return method can also help DDPG avoid divergence during the learning process. Fig.9, which shows a comparison of average learning curves, further indicates the advantage brought about by policy returns. The statistically-derived numbers of trials that are required for various target scores are listed in Tab.5, and these show that the required number of trials can be reduced by 0.06 to 0.58 based on different target scores. From this experiment, we can conclude that the policy return method has wide applicability to deep reinforcement learning algorithms.

Figure 9. Comparison between the performances of DDPG and PR-DDPG on the BipedalWalker-v2 task.

Table 5. Statistical Results of Comparative Verification Experiments for DDPG and PR-DDPG on BipedalWalker-v2.

| Target score | Required number of trials on DDPG | Required number of trials on PR-DDPG |
|--------------|----------------------------------|-------------------------------------|
| 250          | 1.19                             | 1.13                                |
| 260          | 1.38                             | 1.35                                |
| 270          | 1.56                             | 1.47                                |
| 280          | 1.61                             | 1.56                                |
| 285          | 1.85                             | 1.67                                |
| 290          | 2.17                             | 1.78                                |
| 295          | 2.50                             | 1.92                                |

D. EXPERIMENT 4: COMPARISON OF THE POLICY RETURN METHOD WITH OTHER STATE-OF-THE-ART ALGORITHMS

Improving the stability of the learning is one of the most effective ways to reduce the number of trials in DRL.
Experience replay [2] and distributed learning [16] are two state-of-the-art methods with regard to improving the stability of DRL. In this experiment, we set a few target scores at different levels and applied these two state-of-the-art methods and the policy return method to a DRL with an actor-critic structure to challenge these target scores. For every method, the number of trials needed to successfully achieve each target score was recorded.

**Environment:** This experiment conducts these comparisons on the Pendulum-v0 and BipedalWalker-v2 tasks.

**Experimental Details:** Every method was tested 10 times on each target score, and the average required numbers of trials were compared. For the experience replay method, the capacity of the experience memory was 50000 and 200000 for Pendulum-v0 and BipedalWalker-v2, respectively, and a prioritized sampling method [31] was used when choosing transitions. For the distributed learning method, we used 5 agents to learn in the respective environment. For the policy return method, the inspection period $C$ was 200 episodes for Pendulum-v0 and 100 episodes for BipedalWalker-v2. The details of the actor-critic structure are listed in Tab.6.

**Results and Discussion:** As Fig.10 shows, all methods required more trials when the target score was set higher, and in both environments, the policy return method used the fewest trials to overcome every target score. In the Pendulum-v0 trials, when setting the target score higher than $-180$, the policy return method brought about a reduction in the number of required trials of more than 1 compared with those of the experience replay and distributed learning methods. In the BipedalWalker-v2 trials, the policy return method obtained a score of 290 with an average of 1.9 attempts, and the experience replay, which achieved the better performance of these two state-of-the-art methods needed approximately 2.6 attempts.

**V. CONCLUSION**

In this work, we have proposed the policy return method to reduce the number of trials required in deep reinforcement learning. Our method monitors the learning performance in real time and compels the learned policy to return to the current optimal policy when the learning falls into a terrible situation. The experimental results demonstrate that, the policy return method can reduce the number of trials required by existing DRL algorithms by 10 to 40 percent, and it can achieve a fairly good score with nearly one fewer trial than the state-of-the-art algorithms. A drawback of our method that is worthy of discussion is that for the policy return method, a copy of the networks is needed to store the optimal policy; as a result, more GPU memory is required, but this issue will become increasingly negligible as graphics cards are upgraded. Although the policy return method is not impeccable, it can provide practical help, and we hope it can become a useful component for DRL architectures.

**VI. CODE AVAILABILITY**

The source-code for policy return method is available from https://github.com/Parker1988/Experiments

### TABLE 6. Detailed Configurations of the Actor-Critic Structure.

| Hyperparameter | Value                  |
|----------------|------------------------|
| structure (actor) | 256, 64 (relu, relu) |
| structure (critic) | 500 (relu)            |
| learning rate (actor) | 5^-4               |
| learning rate (critic) | 8^-4                |
| discount factor $\gamma$ | 0.93                |
| batch size | 32                     |

**FIGURE 10.** Comparisons between the required numbers of trials by the policy return method and the state-of-the-art methods: (a) the comparison on Pendulum-v0, (b) the comparison on BipedalWalker-v2.
REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with deep reinforcement learning,” 2013. arXiv:1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey.” IEEE Signal Proces. Mag., vol. 34, no. 6, pp. 26–38, Nov. 2017.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grawe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of go without human knowledge,” Nature, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” 2015. arXiv:1504.00702. [Online]. Available: http://arxiv.org/abs/1504.00702

[6] A. Sallab, M. Abdou, E. Perot, and S. Yogananami, “Deep reinforcement learning framework for autonomous driving,” Electron. Imag., vol. 2017, no. 19, pp. 70–76, Jan. 2017.

[7] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in Proc. IEEE Int. Conf. Robot. Automat. (ICRA), May 2017, pp. 3389–3396.

[8] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Sep. 2017, pp. 31–36.

[9] H. Mao, “Resource management with deep reinforcement learning,” in Proc. 15th ACM Workshop Hot Topics Netw., 2016, pp. 50–56.

[10] L. Zhang, J. Tan, Y.-C. Liang, G. Feng, and D. Niyato, “Deep reinforcement learning for modulation and coding scheme selection in cognitive HetNets,” in Proc. IEEE Int. Conf. Commun. (ICC), Shanghai, China, May 2019, pp. 1–6.

[11] X. Wang, J. Wang, Y. Xu, J. Chen, L. Jia, X. Liu, and Y. Yang, “Dynamic spectrum anti-jamming communications: Challenges and opportunities,” IEEE Commun. Mag., vol. 58, no. 2, pp. 79–85, Feb. 2020.

[12] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” 2017, arXiv:1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in Proc. Int. Conf. Mach. Learn., 2015, pp. 1889–1897.

[14] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, arXiv:1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2015, arXiv:1511.05952. [Online]. Available: http://arxiv.org/abs/1511.05952

FENG LIU was born in Banzhou, Hebei, China, in 1988. He received the B.S. degree in mechanical engineering from the Hebei University of Science and Technology, Shijiazhuang, China, in 2011, and the M.S. degree in aeronautical engineering from Nanchang Hangkong University, Nanchang, China, in 2013. He is currently pursuing the Ph.D. degree with the State Key Laboratory of VR Technology and Systems, Beihang University (BUAA). His current research interests include deep learning, reinforcement learning, the autopilot of aircraft, and image recognition.

SHULING DAI received the Ph.D. degree in automatic control engineering from Beihang University (BUAA), Beijing, China, in 1997. He is currently a Professor of the State Key Laboratory of VR Technology and Systems, Beihang University, where he is also a Researcher of the Jiangxi Research Institute. His current research interests include machine learning, VR technology, automatic control engineering, and flight simulation.

YONGJIA ZHAO received the B.S. degree in aircraft design engineering and the Ph.D. degree in automatic control engineering from Beihang University (BUAA), Beijing, China, in 2002 and 2010, respectively. He is currently a Lecturer of the School of Automation Science and Electrical Engineering, Beihang University, where he is also a Researcher of the State Key Laboratory of VR Technology and Systems. His current research interests include deep learning, virtual reality, and flight simulation.