Hindawi
Complexity
Volume 2021, Article ID 6658724, 16 pages
https://doi.org/10.1155/2021/6658724

Research Article

Averaged Soft Actor-Critic for Deep Reinforcement Learning

Feng Ding, Guanfeng Ma, Zhikui Chen, Jing Gao, and Peng Li

School of Software Technology, Dalian University of Technology, Dalian, China

Correspondence should be addressed to Jing Gao; gaojinghit@gmail.com

Received 16 November 2020; Revised 1 February 2021; Accepted 18 March 2021; Published 1 April 2021

Academic Editor: Ning Cai

With the advent of the era of artificial intelligence, deep reinforcement learning (DRL) has achieved unprecedented success in high-dimensional and large-scale artificial intelligence tasks. However, the insecurity and instability of the DRL algorithm have an important impact on its performance. The Soft Actor-Critic (SAC) algorithm uses advanced functions to update the policy and value network to alleviate some of these problems. However, SAC still has some problems. In order to reduce the error caused by the overestimation of SAC, we propose a new SAC algorithm called Averaged-SAC. By averaging the previously learned action-state estimates, it reduces the overestimation problem of soft Q-learning, thereby contributing to a more stable training process and improving performance. We evaluate the performance of Averaged-SAC through some games in the MuJoCo environment. The experimental results show that the Averaged-SAC algorithm effectively improves the performance of the SAC algorithm and the stability of the training process.

1. Introduction

To generate fully autonomous agents which can learn to automate behaviors by interacting with the experimental environment via trials and errors is one of the most important tasks in the current artificial intelligence field. In the current artificial intelligence field, the long-term challenge is to create an intelligence system responding to environments in a timely manner. Such intelligence systems include robots that can interact with the surrounding environments and software-based agents that can interact with multimedia devices. Currently, deep reinforcement learning (DRL) whose mathematical framework is the experience-driven autonomous learning is the most important algorithm to address these challenges [1]. For example, the Google AlphaGo defeated the world champion in the Go game. To complete the breakthrough in this field, there are still a lot of work to do. Among them, ensuring the safety of decision making is one of the most important challenges. Because the agent is more inclined to explore unfamiliar states, deep reinforcement learning will be susceptible to the so-called security exploration problem, causing the agent to be in an unsafe state (for example, a mobile robot drove the car into the ditch). The safe intelligence system should ensure the safety of the controlled objects and reduce the probability of dangerous operations. Since 2018, a series of unmanned vehicle safety accidents have occurred in the United States, highlighting the importance of safety in automatic control tasks. However, many current artificial intelligence methods do not fully control risks. Furthermore, some methods deliberately add exploratory learning with random nature to the solution process. The exploratory learning without security restrictions is likely to bring risks. If the agent directly applies the reinforcement learning method for "trial and error" exploration and learning in real-world tasks, the decision made by the agent may put the system into a dangerous state. The security of deep reinforcement learning has attracted more and more attention. Therefore, to improve the safety and address the noncontrol problems in practical applications, we should find ways to reduce the propagation error in the neural network. At the same time, the stability of the DRL algorithm is a big challenge, which limits the further development of the algorithm. Although the performance of the computer has been greatly improved, the stability of DRL cannot be guaranteed. Therefore, how to improve the security and stability of DRL algorithms for a large number of artificial intelligence learning is one of the most challenging problems in DRL today.

Nowadays, the Deep Q-Network algorithm (DQN) [2, 3], as the reinforcement learning extension, shows good results in dealing with some complex high-dimensional image information tasks. DQN combines the Q-learning of reinforcement learning with the convolutional neural network (CNN), which learns the control policy directly from the high-dimensional raw pixels of games. In this way, DQN can reach human level scores in certain Atari games. After that, many DQN-based models have been proposed to enhance the learning ability and stability. Among these extensions, the Soft Actor-Critic (SAC) algorithm [4] is a well-known highly optimized version. The Soft Actor-Critic algorithm is an off-policy Q-learning algorithm based on maximum entropy. Its main advantages are high sampling efficiency and robustness by using the stochastic policy. Soft Actor-Critic has achieved good results in public benchmark tests and can be directly applied to real robots.

1.1. Motivation. The current SAC algorithm has problems of insecurity and poor stability during the training process. Specially, the SAC algorithm is an off-policy Q-learning algorithm based on maximum entropy, which uses the experience replay mechanism [5]. Furthermore, it is easy to overestimate states for SAC by using the max operation to quickly move the Q-value closer to the possible optimization target [6–8]. Therefore, the motivation of this paper will focus on how to improve the security and stability of the SAC algorithm.

1.2. Challenge. To improve the performance of the SAC algorithm, we need to meet some challenges in implementing the new algorithm. In particular, the overestimation of the soft Q-value will lead to higher deviations which are caused by the actions that cannot be sampled in SAC [9]. This deviation causes the algorithm to require more sample data during the training process and to converge to a bad situation that is not a local optimum [10]. Therefore, we discuss the main challenges from the following aspects: (1) to improve the stability and convergence of the SAC algorithm, we need to know the reason why the SAC algorithm is not so stable during the training process; (2) to prove the superiority of our proposed algorithm, we need to conduct experimental comparisons in different game environments.

1.3. Contribution. We propose an Averaged-SAC algorithm. Our work is based on the average of the previously known Q-value estimates [11], using the average soft Q-value to solve the problem caused by the combination of the Q-learning and function approximation problem. This improved algorithm reduces the variance of the target approximation error and improves the stability of the algorithm training process. After using this algorithm, the learning ability of the agent is greatly improved, thereby improving the performance of the model. The contributions in this paper are listed as follows:

(1) We analyzed the reasons why the SAC algorithm is sometimes unstable. Through theoretical analysis, we found that the soft Q-value of SAC has an overestimation problem, which also causes SAC to converge to a bad situation that is not local optimal.

(2) We propose the Averaged-SAC algorithm. To deal with the problem caused by overestimation of the soft Q-value in the SAC algorithm, our work solves the overestimation phenomenon by the average of the previously known Q-value estimates. This proposed algorithm reduces the variance of the target approximation error and improves the stability of the algorithm training process, which improves the learning ability of the agent.

This paper is arranged as follows. Section 2 introduces the preliminary knowledge related to the paper. Section 3 explains some basic knowledge of deep reinforcement learning and SAC algorithms. Section 4 mainly introduces the proposed algorithm, and Section 5 describes the experimental results of some of our continuous tasks in MuJoCo. Finally, Section 6 summarizes the contribution of this paper.

2. Preliminaries

As a very important field of machine learning, reinforcement learning aims to find the best policy for sequential decision problems through learning. Q-learning can optimally solve the Markov decision process. However, Q-learning also has its shortcomings. In the process of learning policy, Q-learning algorithm [12, 13] includes the step of maximizing Q-value, which causes it to overestimate the action value during the learning process. In order to avoid this overestimation, researchers proposed double Q-learning and double deep Q-networks later to achieve lower variance and higher stability [7]. The solution idea of Double-DQN algorithm (DDQN) is to provide a method to reduce the chance of overoptimistic estimation. And this method prevents overfitting.

However, because DDQN uses the previous step of Q-value for training, it has the same problem as DQN. The change of DQN makes the training process unstable. This situation tells us that even if only one step of Q-value is taken during the training process, DQN will still be unstable. Later, in order to solve the problems caused by this Q-learning algorithm, researchers proposed the Averaged-DQN. The algorithm uses K previously learned Q-values to estimate the current action value. By using the average Q-value of the first K steps for training, the variance introduced by Q-learning can be reduced, and the approximation error [14] can be reduced, and thus it stabilizes the entire training process. Double-DQN is an algorithm that uses two sets of deep neural networks: one for selecting actions and the other for evaluating actions. DDQN prevents excessive valuation. In short, DDQN separates selection from evaluation to prevent overestimation.

The Averaged-DQN (ADQN) is a simple extension of the DQN algorithm, which uses K previously learned Q-values to generate new action values. Researchers claim that ADQN can be easily integrated with other DQN or

DDQN. The ADQN uses the previous K learned Q-values to generate the current Q-value estimate to reduce the variance caused by the training process.

Compared with DQN, DDQN theoretically reduces excessive optimism by decomposing the maximum operation into action selection and action evaluation, thereby improving performance. Compared with DQN, ADQN can not only solve the overestimation problem but also reduce the variance of the approximate error of the target value. Therefore, the average maximum Q-value of ADQN should also be less than DQN. In addition, the performance of ADQN should be better than DQN, which means that the average score of each episode of ADQN is theoretically greater than DQN. In addition, compared with DQN and DDQN, it should have better convergence, and the performance lines of the agent are smoother and more stable.

The SAC algorithm, one of the most popular Actor-Critic (AC) algorithms [15], also overestimates the soft Q-value estimation during the training process. And because SAC only uses the previous step of soft Q-value for calculation, it has the same problem as the DQN algorithm. That is, SAC also has the phenomenon of unstable training process. In order to solve this problem, we innovatively apply the idea of average Q-value to the calculation of soft Q-value on Actor-Critic in the SAC model. We propose the Averaged-SAC algorithm. In order to cope with the problems caused by overestimation of the soft Q-value in the SAC algorithm, our work is based on the average of the previously known Q-value estimates [11] and proposes to solve the overestimation phenomenon. This solution is the Averaged-SAC. This improved algorithm reduces the variance of the target approximation error and improves the stability of the algorithm training process. After using this algorithm, the learning ability of the agent is greatly improved, thereby improving the performance of the model.

3. Background

In this section, we will introduce the relevant background knowledge of Actor-Critic (AC) and Soft Actor-Critic (SAC).

Machine learning is developing rapidly, and deep reinforcement learning (DRL), especially DRL algorithm for continuous control, plays an important role [12, 14]. DRL methods can be divided into model-based DRL methods and model-free DRL methods. Among them, the model-free DRL methods include action-value approximation (value-based) methods and policy gradient (policy-based) methods. The action-value approximation methods use the estimation of action-value to update the policy [13]. As for policy gradient method, Actor-Critic (AC) algorithm is an important example.

3.1. Actor-Critic.

The current research finds that the policy gradient method is better than the Q-learning method when solving the optimal policy selection problem in DRL. So, in recent years, policy gradient (PG) methods have been applied in many areas. The Actor-Critic method is a combination of policy-based and value-based method because it is a PG method and combines with the value estimation method. However, it is usually classified as a PG method [16] in some places.

In the Actor-Critic framework [16], the actor is responsible for the policy gradient learning, while the critic is responsible for the policy evaluation to estimate the value function. It can be seen that, on the one hand, the actor learns the policy, and the policy update depends on the value function estimated by the critic; on the other hand, the critic estimates the value function, and the value function is the function of the policy learned by the actor. Policy and value function depend on each other and affect each other, so iterative optimization is needed during the training process. This iterative idea of multivariate optimization is actually reflected in machine learning.

The Actor-Critic model is a better scoring function. Different from Monte Carlo method, its method of updating parameters is the Temporal-Difference Learning (TD) method. The parameters of the Actor-Critic model are updated immediately after each step of training is completed, so there is no need to wait until the end of the episode. The update of θ which is the parameter of the actor network is as follows:

$$\Delta\theta = \alpha\nabla_\theta\left(\log \pi_\theta(s,a)\right)\widehat{q}_w(s,a), \tag{1}$$

where s is the current state, a is the current action, α is policy learning rate, and w is the parameter of the critic network. $\widehat{q}_w(s,a)$ is the critic, Q-learning function approximation (estimate action-value) which calculates the score q value for action a in the current state s; $\pi_\theta(s,a)$ is the actor, policy function which uses this q value to update its policy weight.

The update of w is as follows:

$$\Delta w = \beta\left(R(s,a) + \gamma\widehat{q}_w(s_{t+1},a_{t+1}) - \widehat{q}_w(s_t,a_t)\right)\nabla_w\widehat{q}_w(s_t,a_t), \tag{2}$$

where $R(s,a)$ is the current cumulative reward, β is the policy learning rate, γ is the value learning rate, $(R(s,a) + \gamma\,\widehat{q}_w(s_{t+1},a_{t+1}) - \widehat{q}_w(s_t,a_t))$ is TD-error, and $\nabla_w\widehat{q}_w(s_t,a_t)$ is the gradient of the value function.

The Actor-Critic algorithm was improved later, and the improved algorithm is called "Advantageous Actor-Critic" (A2C). A2C uses an advantage function instead of the original reward in the critic network, which can be used as an indicator to measure the value of selected actions and the average of all actions. A baseline is added to the Q-value to judge the feedback positive or negative. The baseline here is usually represented by the value function of the state, so the update of the parameter of the critic network w is defined as shown below:

$$\Delta w = \beta\left(R(s,a) + \gamma\widehat{q}_w(s_{t+1},a_{t+1}) - V(s)\right)\nabla_w\widehat{q}_w(s_t,a_t), \tag{3}$$

where $(R(s,a) + \gamma\widehat{q}_w(s_{t+1},a_{t+1}))$ is $Q(s,a)$, $s_t \in S$ is the state at time step t, $a_t \in A$ is the execution action at time step t, $V(s)$ is the average value of state s, and $A(s,a) = Q(s,a) - V(s)$ is called the advantage function. The advantage function uses the action-value function to subtract the baseline value owned by its corresponding state, so that the result of this calculation becomes the gain brought by the action, thereby reducing the variance caused by the change of the state baseline value.

The Proximal Policy Optimization (PPO) algorithm [17] is currently the mainstream DRL algorithm. It faces both discrete control and continuous control, and it has achieved great success in OpenAI. However, PPO is an on-policy algorithm, that is, PPO is faced with serious sample inefficiency. Because PPO [17, 18] needs to resample enough samples under the current policy to update each policy, the loss caused by such an update method is very large. Therefore, the previous sampled data are completely discarded, and the large number of samples make the algorithm complexity too high to ensure convergence. This leads to the need to learn a large number of samples, which is unacceptable for real robot training.

Deep Deterministic Policy Gradient (DDPG) [16] (and its extension) is an off-policy algorithm, which is more sample efficient than PPO. DDPG trains a deterministic policy, that is, only one optimal action is considered at each state. However, off-policy methods such as DDPG also have shortcomings. When reinforcement learning learns from low-dimensional feature vectors, different variables of different observations may have different physical units (such as speed and position), so their value ranges will be very different, which may cause the network to fail to learn effectively and determine the parameters of the top-level network architecture. DDPG manually scales the function so that there will be similar value ranges. The method used by DDPG is batch normalization in deep learning, to normalize the data of each dimension to be with zero mean and unit variance. And during operation, it continues to calculate the floating average to maintain normality. The characteristics of DDPG result in a large amount of calculation and the difficulty to stabilize convergence. On the one hand, the DDPG algorithm is extremely sensitive to the setting of hyperparameters, and the convergence is difficult to stabilize. In order to make the loss converge to a good level, the DDPG algorithm often needs to try to adjust the hyperparameters many times. On the other hand, although the use of Replay Buffer solves the problem of sample utilization efficiency, the policy and Q-value are coupled with each other, which makes its performance unstable and the final score obtained by the DDPG algorithm often very low.

Compared with DDPG, SAC [19] uses stochastic policy, which has certain advantages over deterministic policies. SAC has achieved very good results in the public benchmarks and can be directly applied to real robots.

3.2. Soft Actor-Critic. The goal of SAC is to maximize the reward obtained from the interaction between agent with the environment. To achieve this goal, SAC uses soft policy iteration. The process of soft policy iteration is to alternate policy evaluation and policy improvement within the framework of maximum entropy. Experimental results show that, in terms of learning speed and robustness, SAC consistently outperforms other RL algorithms in continuous action benchmark tests. SAC tries to find a policy to maximize the maximum entropy goal.

The SAC has several components in the iterative process: the policy network ϕ which is responsible for receiving the state of the current environment and calculating the average value and standard deviation of the state's action distribution, the soft Q-network θ which is used to estimate the value of the state action pair, the value network ψ for evaluating the state value, and the target value network $\overline{\psi}$ which is just the exponential moving average of the value network ψ. In the original SAC, only one sample was collected during the data collection (one interaction with the environment), and a small batch update was performed during the update. In the implementation of our algorithm model, we determine that the first task is to collect the data of the episode until the episode is terminated. There are two conditions for episode termination: improper operation or the agent has completed the predefined number of steps per episode; here we set it to 1000. Then, we choose to update each batch in order to update the data better. The number of updates is set to be the same value as the length of the model episode.

Because current versions of SAC do not give a good solution for reducing the overestimation of the soft Q-value, we propose an Averaged-SAC to solve the bias problem caused by the overestimation of the soft Q-value.

4. Averaged Soft Actor-Critic

We first analyzed the causes of soft Q-value in SAC, which made a good theoretical analysis for our method. Although SAC uses the maximization operation to quickly move the Q-value close to the possible optimization goal, it is easy to overestimate. SAC's soft Q-value has the problem of overestimation which will result in an "excessive" estimated sampled action. For some actions that are not sampled in the SAC, these actions will not be selected as the optimal action. Therefore, it is easy to overestimate the Q-value in SAC. SAC will tend to choose the current action with the best performance, although this action is overestimated. At the same time, SAC is based on sampling after all, so there will be some states in the interactive environment of SAC that are not sampled. These unsampled states will affect the learning ability of the agent, resulting in bias. Therefore, in view of the problems caused by overestimating the soft Q-value of the SAC algorithm, we have carried out an innovative extension to the SAC algorithm and proposed the Averaged-SAC algorithm. This improved algorithm reduces the variance of the target approximation error, thereby improving the stability of training and the performance of the agent.

The Averaged-SAC tries to find a policy to maximize the maximum entropy goal:

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} E_{(s_t,a_t)\sim\tau_\pi}\left[\gamma^t\left(r\left(s_t,a_t\right)+\alpha\mathscr{H}\left(\pi\left(\cdot|s_t\right)\right)\right)\right], \tag{4}$$

where π is the agent's policy, π^* is the agent's optimal policy, T is the total number of steps in the agent's action time,

r: $S \times A \longrightarrow \mathbb{R}$ is the reward function of Averaged-SAC, $s_t \in S$ is the state at time step t, $a_t \in A$ is the executed action at time step t, and $\gamma \in [0,1]$ is the discount rate. Here, τ_π is the state action boundary of the trajectory distribution caused by π. Hyperparameter α balances exploration and exploitation of reinforcement learning and affects the randomness of optimal policies. Because this hyperparameter determines the relative importance of the entropy term with respect to the reward, it is called the temperature parameter. $\mathscr{H}(\pi(\cdot|s_t)$ is the entropy of the policy π in the state s_t, and its calculation formula is $\mathscr{H}(\pi(\cdot|s_t)) = -\log \pi(\cdot|s_t)$.

Policy evaluation is the first step of the Averaged-SAC's soft policy iteration. Policy evaluation needs to calculate the value of policy π. To achieve this goal, the Averaged-SAC defines the soft state-value function as

$$V(s_t) := E_{a_t \sim \pi}[Q(s_t,a_t) - \alpha \log(\pi(a_t|s_t))], \tag{5}$$

where π is the agent's policy, $s_t \in S$ is the state at time step t, an $a_t \in A$ is the executed action at time step t.

The soft Q-function of Averaged-SAC is defined as

$$T^\pi Q(s_t,a_t) := r(s_t,a_t) + \gamma E_{s_{t+1 \sim p(s_t,a_t)}}[V(s_{t+1})], \tag{6}$$

where p: $S \times A \longrightarrow S$ is the distribution of states. Agent can get the next state according to the current state and action.

The state value function approximates the soft value. There is no need in principle to include a separate function approximator for the state value, since it is related to the Q-function and policy according to equation (5). This quantity can be estimated from a single action sample from the current policy without introducing a bias, but in practice, including a separate function approximator for the soft value can stabilize training and is convenient to train simultaneously with the other networks. The soft value function is trained to minimize the squared residual error:

$$J_V(\psi) = \mathbb{E}_{S_t \sim \mathscr{D}}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}\left[Q_\theta(s_t,a_t) - \log \pi_\phi(a_t|s_t)\right]\right)^2\right], \tag{7}$$

where D is the distribution of previously sampled states or a replay buffer. The gradient of equation (7) can be estimated with an unbiased estimator:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)\left(V_\psi(s_t) - Q_\theta(s_t,a_t) + \log \pi_\phi(a_t|s_t)\right), \tag{8}$$

where the actions are sampled according to the current policy, instead of the replay buffer.

When SAC is applied to the continuous state space, the neural network with parameter θ of the SAC algorithm needs to be updated. The SAC algorithm minimizes the soft Bellman residual by training the soft Q-function:

$$J_Q(\theta) = E_{(s_t,a_t)\sim D}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \left(r(s_t,a_t) + \gamma \mathrm{E}_{s_{t+1} \sim p(s_t,a_t)}\left[V_{\overline{\psi}}(s_{t+1})\right]\right)\right)^2\right], \tag{9}$$

where D is the replay buffer of past experience, $s_t \in S$ is the state at time step t, $a_t \in A$ is the executed action at time step t, p: $S \times A \longrightarrow S$ is the distribution of states, $\overline{\psi}$ is the target value network which is just the exponential moving average of the value network ψ, and θ is the soft Q-network which is used to estimate the value of the state action pair. Averaged-SAC uses the Q-value of the target network and the Monte Carlo estimation to estimate the residual.

The Averaged-SAC algorithm selects the K previously learned soft Q-values before the current step in the process of interaction between the agent and the environment to calculate the average soft Q-value. Therefore, the average soft Q-value calculation formula is as follows:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim \mathscr{D}}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \left(r(s_t,a_t) + \gamma \frac{1}{K}\sum_{k=1}^{K} V_{\overline{\psi}}(s_{t+1})\right)\right)^2\right]. \tag{10}$$

Through the comparison between Averaged-SAC and SAC in the InvertedDoublePendulum game in Figure 1, it can be seen that the proposed Averaged-SAC algorithm has a significant effect on reducing the overestimation of soft Q-value. Therefore, it is proved that the proposed algorithm can greatly reduce the overestimation problem.

The principle of the policy improvement step is to update the parameters as far as possible in the direction of maximizing the reward that the agent can obtain. The Averaged-SAC algorithm updates the temperature to the Boltzmann policy, where the temperature is α and the Q-function is used as energy. Therefore, the Averaged-SAC algorithm updates the policy by minimizing the Kullback–Leibler (KL) divergence between it and the Boltzmann policy. The policy improvement formula is as follows:

$$\pi_{\mathrm{new}} = \underset{\pi \in \prod}{\arg\min}\, D_{KL}\left(\pi(\cdot|s_t)\middle\|\frac{\exp(1/\alpha Q^\pi \mathrm{old}(s_t,\cdot))}{Z^\pi \mathrm{old}(s_t)}\right), \tag{11}$$

where the partition function $Z^{\pi_{\mathrm{old}}}(s_t)$ is difficult to handle but not helpful in improving the new policy, so it can be

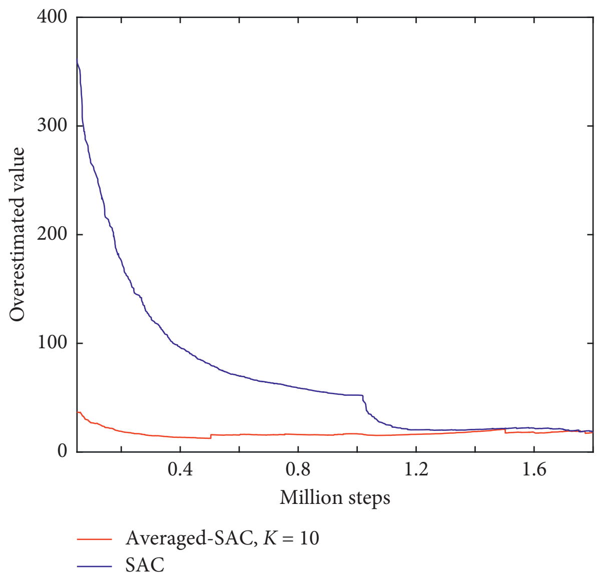


FIGURE 1: Results of the overestimation of Averaged-SAC and SAC.

ignored. The KL divergence of the above formula needs to be minimized to learn the agent's policy network parameters. The minimized KL divergence means the expected value of the reverse maximization state-action value function, so that a more accurate regular term of the entropy of the action distribution can be captured.

For the policy $\pi_\phi(a_t|s_t)$, the first task of Averaged-SAC is to use neural network ϕ to parameterize it. In the process of parameterized training for continuous states and continuous actions, the SAC neural network ϕ will output two values: one is the mean and the other is the covariance. These two values are used to define the Gaussian policy distribution. In order to minimize the expected KL divergence formula, the SAC algorithm calls the temperature parameter α and multiplies the temperature parameter α by the expected KL divergence while ignoring the distribution function. After minimizing the expected KL divergence, the gradient is not affected any longer, so the purpose of learning and updating policy parameters is achieved:

$$J_\pi(\phi) = E_{s_t \sim D}\left[E_{a_t \sim \pi_\phi}\left[\alpha \log\left(\pi_\phi(a_t|s_t)\right) - Q_\theta(s_t, a_t)\right]\right], \tag{12}$$

where D is the replay buffer of past experience, $s_t \in S$ is the state at time step t, $a_t \in A$ is the executed action at time step t, θ is the soft Q-network which is used to estimate the value of the state action pair, and ϕ is the policy network.

As we all know, the backpropagation of neural networks is an important way to update parameters. Because SAC has taken an expectation over the distribution of policy output actions, it leads to error transmitted wrongly in backpropagation. The SAC algorithm uses a reparameterization technique [20] to solve the problem. SAC combines its corresponding neural network output with the input noise vector sampled from a spherical Gaussian to establish that the backpropagation error is correctly transmitted. In this way, the output of the policy network can be used to form a random action distribution without directly using the output of the policy network. The following formula indicates that SAC reparameterizes the policy by the above reparameterization technique:

$$a_t = f_\phi(\epsilon_t; s_t), \tag{13}$$

where $\epsilon_t \sim N(0, I)$. After this change, the policy objective will become

$$J_\pi(\phi) = E_{s_t \sim D, \epsilon_t \sim N}\left[\alpha \log\left(\pi_\phi\left(f_\phi(\epsilon_t; s_t)|s_t\right)\right) - Q_\theta\left(s_t, f_\phi(\epsilon_t; s_t)\right)\right], \tag{14}$$

where the policy f_ϕ is now implicitly defined according to the formula of f_ϕ.

However, the ultimate goal of the temperature parameters they obtain is relevant. Learning the temperature parameters of SAC is also a relatively important work of the current SAC algorithm. Haarnoja et al. [19] proposed a new idea to learn temperature parameters. The temperature parameters here cannot be set as hyperparameters because they are constantly changing. Since the details are not strictly related to the proposed Averaged-SAC, we will not repeat them here. The specific formula is as follows:

$$J(\alpha) = E_{a_t \sim \pi_t}\left[-\alpha\left(\log \pi_t(a_t|s_t) + \overline{H}\right)\right], \tag{15}$$

where $\overline{H}$ is a constant vector equal to the hyperparameter, representing the entropy of the target.

In Algorithm 1, the Averaged-SAC algorithm, a variant of SAC, is introduced.

We retain K previously learned parameters of the value network during the training process. From equations (9) and (10), we can see the difference between the traditional SAC algorithm and the Averaged-SAC in the soft Q-value network parameter calculation. In the process of training the SAC algorithm, we use K previously learned state-value estimates to generate an average state value, thereby

```
Initialize: initialize parameter vectors $\psi, \overline{\psi}, \theta, \phi$; initialize average size $K$
 (1) for each iteration do
 (2) for each environment step do
 (3) $a_t \sim \pi_\phi (a_t|s_t)$
 (4) $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 (5) $\mathscr{D} \longleftarrow \mathscr{D} \cup \{(s_t, a_t, t(s_t, a_t), s_t + 1)\}$
 (6) end for
 (7) for each gradient step do
 (8) $\psi \longleftarrow \psi - \lambda_V \widehat{\nabla}_\psi J_V(\psi)$
 (9) $\theta_i \longleftarrow \theta_i - \lambda_Q \widehat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
(10) $\phi \longleftarrow \phi - \lambda_\pi \widehat{\nabla}_\phi J_\pi(\phi)$
(11) $\overline{\psi} \longleftarrow \tau\psi + (1 - \tau)\psi$
(12) end for
(13) end for
```

Algorithm 1: Averaged Soft Actor-Critic (Averaged-SAC).

reducing the overestimation error caused by a single state-value estimate. This operation reduces the variance generated during the propagation of the soft Q-network, so that the convergence of the SAC algorithm becomes stable.

In Algorithm 1, we can see how the parameters of each neural network are updated:

$$\widehat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(S_t)\big(V_\psi(S_t) - Q_\theta(S_t, a_t) + \log \pi_\phi(a_t|S_t)\big), \tag{16}$$

where the actions are sampled according to the current policy, instead of the replay buffer (a feature of this algorithm).

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim D}\left[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \left(r(s_t, a_t) + \gamma \frac{1}{K}\sum_{k=1}^{K} V_{\overline{\psi}}(s_{t+1})\right)\right)^2\right], \tag{17}$$

where D is the replay buffer of past experience, p: $S \times A \longrightarrow S$ is the distribution of states, $\overline{\psi}$ is the target value network which is just the exponential moving average of the value network ψ, and θ is the soft Q-network which is used to estimate the value of the state action pair.

$$J_\pi(\phi) = E_{s_t \sim D, \epsilon_t \sim N}\big[\alpha \log\big(\pi_\phi\big(f_\phi(\epsilon_t; s_t)|s_t\big)\big) - Q_\theta\big(s_t, f_\phi(\epsilon_t; s_t)\big)\big], \tag{18}$$

where the policy f_ϕ is now implicitly defined according to the formula of f_ϕ.

When running the SAC algorithm, we need to train multiple episodes to let the agent learn more knowledge. In each episode of training, SAC algorithm will update the parameters of the networks after each operation or when the terminal state is reached.

The soft Q-value update calculation performed by Averaged-SAC is shown in equation (8). After comparison, it can be concluded that the biggest difference between SAC and Averaged-SAC is that Averaged-SAC uses the average state function to update the soft Q-network. Also, because the SAC algorithm belongs to the Actor-Critic framework, it is a combination of value-based methods and policy-based methods. So, the Averaged-SAC algorithm relies on actor-learner and cumulative updates to improve training stability. At the same time, the update of the policy network parameters and the soft Q-value network parameters are separate, and the CNN network or fully connected network is used by Averaged-SAC as the network architecture. The Actor network finally outputs the policy $\pi_\phi(a_t|s_t)$ through the softmax network layer, and the Critic network linearly outputs the state value $V(a_t|s_t)$ through the fully connected layer, where all nonoutput layers share their parameters.

Compared with the traditional SAC, Averaged-SAC uses the K state values learned by the value network to reduce the overestimation of the soft Q-value, thereby reducing the variance in the parameter transfer process. Therefore, Averaged-SAC can stabilize the training process and improve the performance of the agent.

5. Experiment

This paper uses MuJoCo environment in OpenAI Gym as the experimental environment. Research by Mnih et al. [2] pointed out that in most MuJoCo games, SAC is significantly better than other DRL algorithms. In experiment, we chose 6 MuJoCo games to test the performance of SAC and Averaged-SAC. Averaged-SAC uses K previously learned state value estimates to generate the soft Q-value in SAC. Some used parameters are set as follows: $\tau = 5e - 3$, $\mathrm{lr} = 3e - 4$, and $\alpha = 0.2$. The discount factor γ of all the above algorithms is 0.99. The network architecture used in SAC and Averaged-SAC is the same as that in [19], and the MuJoCo experiments use the same settings as those in [19]. There are three candidate K values in Averaged-SAC: 5, 10, or 20. Finally, we analyzed the stability and performance of the proposed Averaged-SAC algorithm in detail.

The experiment aims to solve the following problems:

(1) Compared with SAC, can the Averaged-SAC improve the performance of learning policies?

(2) How does the number K of average soft Q-values affect the performance of the Averaged-SAC?

5.1. Experimental Environment and Setup. The purpose of our experimental evaluation is to understand how the performance and stability of the learning policy of our method compares with the previous SAC algorithm. We compared our method with existing technologies to solve a series of challenging continuous control tasks and the rllab realization of humanoid robot tasks in the OpenAI gym benchmark suite. Although a variety of different algorithms can be used to solve simpler tasks [21], it is sometimes difficult to achieve stable performance in some tasks using SAC algorithms, such as 21-dimensional Humanoid. Algorithm stability also plays a big role in performance: easier tasks make it more practical to adjust hyperparameters for better results. And for algorithms that are more sensitive on the strictest benchmarks [9], the narrow area of effective hyperparameters becomes very small, resulting in poor performance.

We conducted experiments in a set of MuJoCo environments. We aim to show how the average soft Q-value scheme affects the performance of SAC and how different K values affect the performance of Averaged-SAC. In order to make our experimental comparison more reasonable, we must make the experimental analysis meaningful and the results reproducible. For SAC and Averaged-SAC, we used the same SAC code base implemented on PyTorch for SAC implementation [8].

5.2. Experimental Results of Averaged-SAC and SAC. We tested the performance of Averaged-SAC in the game tasks in the continuous action space. We chose 6 MuJoCo games, including Walker2D, Ant, Hopper, Humanoid, InvertedDoublePendulum, and HalfCheetah. We use these game environments to compare the performance of Averaged-SAC and SAC during training.

The result is shown in Figure 2. It shows that the Averaged-SAC of all 6 games of MuJoCo is better than SAC. For the MuJoCo experiment, each training period (epoch) includes 1000 steps. First, we focus our analysis on the performance of Averaged-SAC (red) compared to the SAC benchmark (blue). For all game environments of Averaged-SAC, $\gamma = 0.99$. The hyperparameter γ is obtained in the SAC.

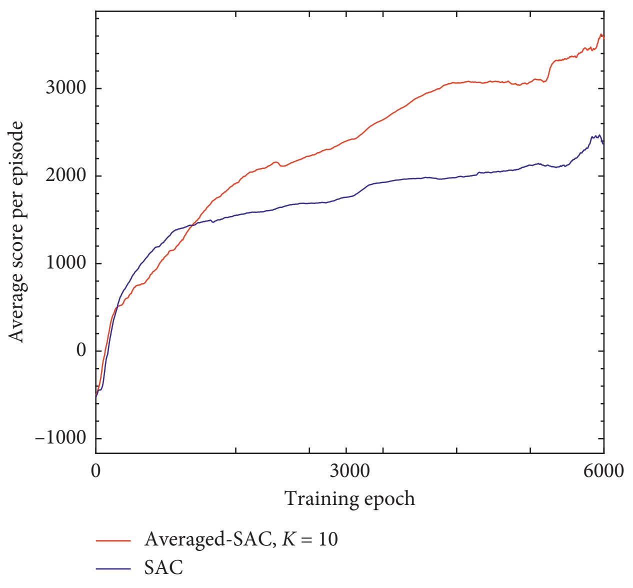
The results show that Averaged-SAC consistently outperforms the SAC benchmark in all environments and all training stages. For example, in Ant-v2, the average performance of Averaged-SAC is 1.5 times that of the benchmark SAC, and the average reward value of Averaged-SAC reaches 2400 score which is 1.8 times faster than SAC; in Hopper-v2, the average reward value of Averaged-SAC reaches 2600 score which is 1.7 times faster than SAC, and the average performance of Averaged-SAC is 1.4 times that of SAC; in Walker2d-v2, the average reward value of Averaged-SAC reaches 5000 score which is 2 times faster than SAC, and the average performance of Averaged-SAC is 2.2 times that of SAC; in HalfCheetah-v2, the average reward value of Averaged-SAC reaches 10000 score which is 1.9 times faster than SAC, and the average performance is 1.7 times that of SAC; in Humanoid-v2, the average reward value of Averaged-SAC is 1.6 times faster than SAC to reach 4300 score, and the average performance of Averaged-SAC is 1.5 times that of SAC; in InvertedDoublePendulum-v2, the average reward value of Averaged-SAC reaches 9000 score which is 1.2 times faster than SAC, and the average performance of Averaged-SAC is 1.3 times that of SAC. In addition, Averaged-SAC is more stable than SAC during training.

5.3. Experimental Results of Averaged-SAC's Loss and SAC's Loss. Figure 3 shows the variation of the loss value of the critic network part in these 6 games. It can be seen from the figure that the Averaged-SAC loss values of all the 6 games of MuJoCo are all lower than those of the SAC during the training process, which indicates that the proposed Averaged-SAC model exactly reduces the error caused by overestimation and greatly helps the soft Q-learning.

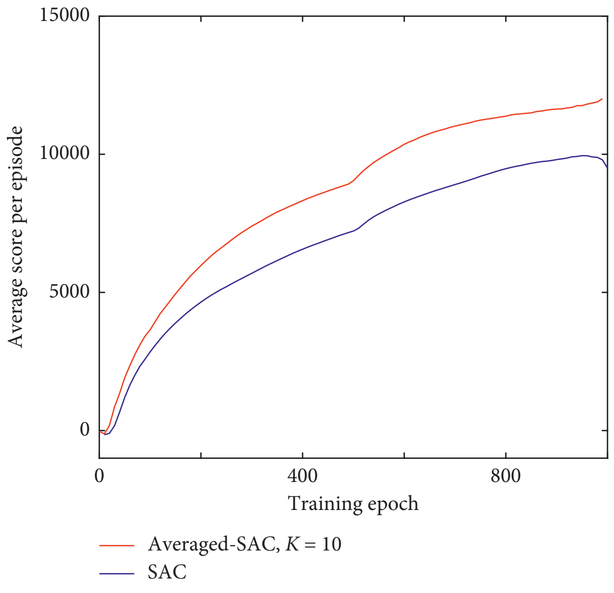
5.4. Experimental Results of Averaged-SAC with Different K Values. Figure 4 shows the performance of Averaged-SAC in 6 different MuJoCo game environments under different K values. It can be seen that when K value is 10, the corresponding Averaged-SAC algorithm is better than other algorithms with different K values. As a result, we can draw the following conclusions: increasing K value within a certain range will result in better performance and stability of Averaged-SAC; if the increased K value range exceeds a certain limit, then the performance of Averaged-SAC and stability will decline.

5.5. Experimental Results of Averaged-SAC's Loss with Different K Values. Figure 5 shows the variation of the loss value of the critic network part of these 6 games under different K values. From the figure, we can see the changes in the Averaged-SAC loss values of all 6 MuJoCo games during the training process. When the K value is 10, the corresponding Averaged-SAC loss value is lower than that of other algorithms with different K values. As a result, we can draw the following conclusion: increasing K value within a certain range will result in a large reduction in the propagation error of Averaged-SAC, further improving the stability of the algorithm.

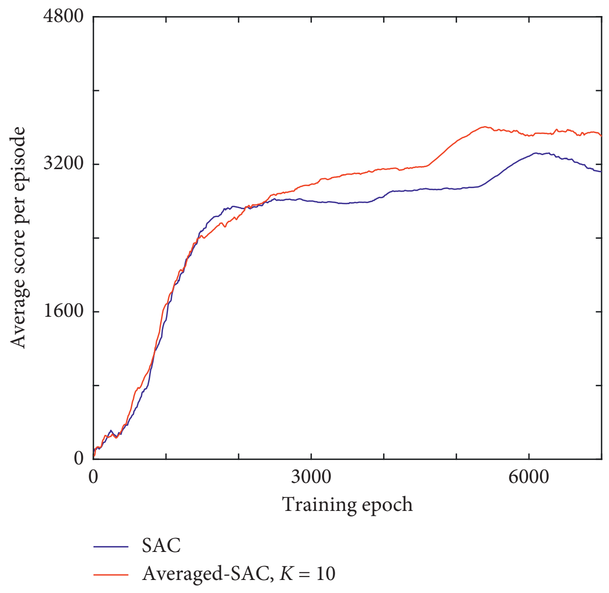
In addition, in order to compare the performance of Averaged-SAC and SAC more intuitively, we use a table to list the average training scores of them. The results are summarized in Table 1. It can be seen that the overall performance of Averaged-SAC is better than that of SAC. In



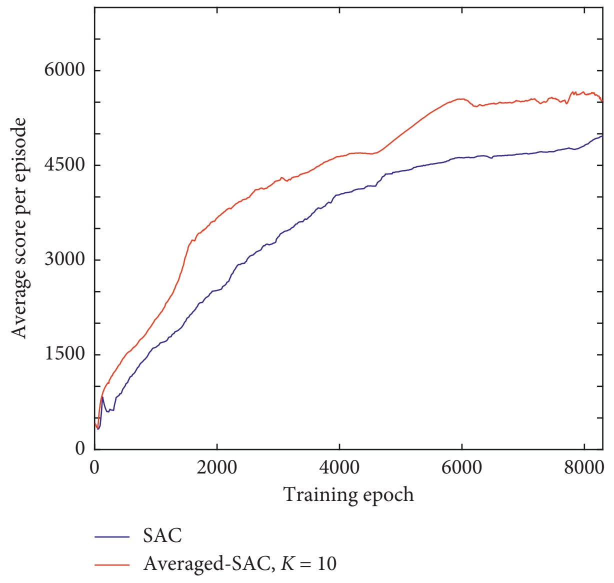
(a)



(b)

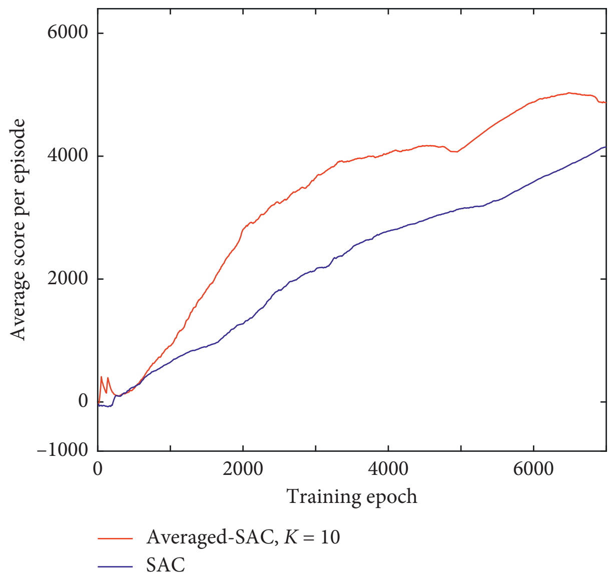


(c)

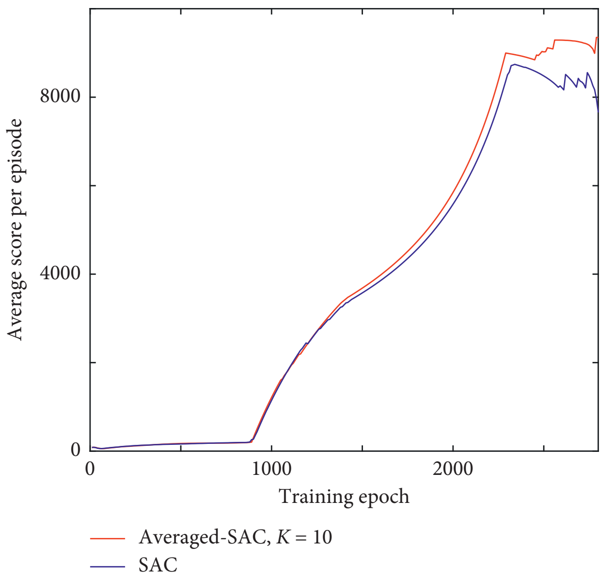


(d)

Figure 2: Continued.

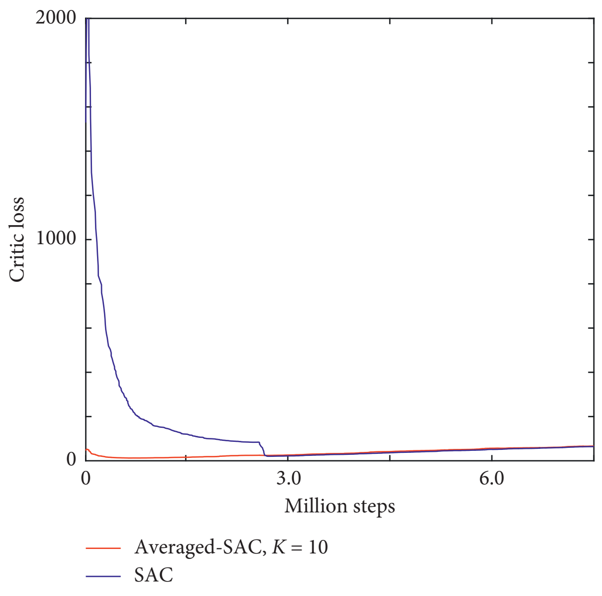


(e)

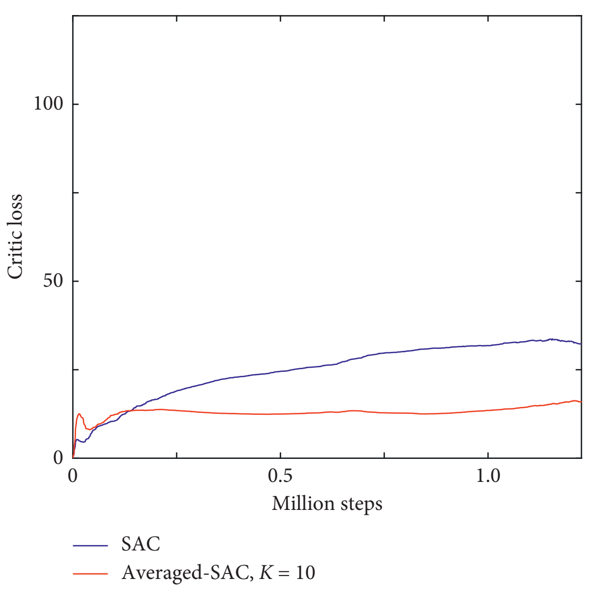


(f)

Figure 2: Results of Averaged-SAC and SAC in 6 MuJoCo games. (a) Ant-v2. (b) HalfCheetah-v2. (c) Hopper-v2. (d) Humanoid-v2. (e) Walker2D-v2. (f) InvertedDoublePendulum-v2.

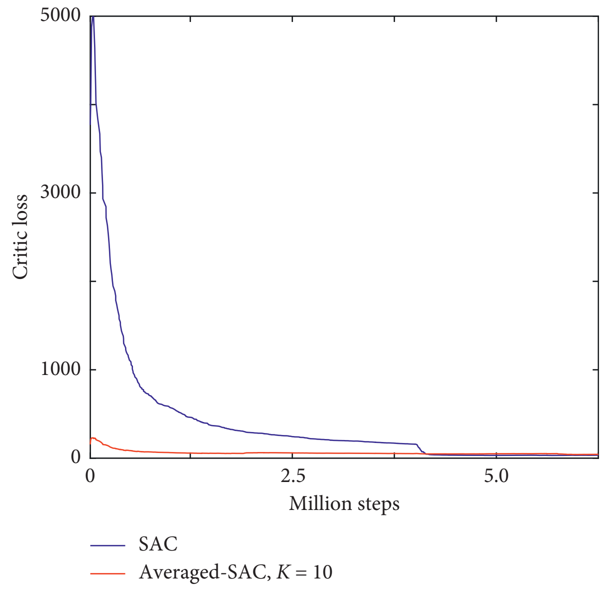


(a)

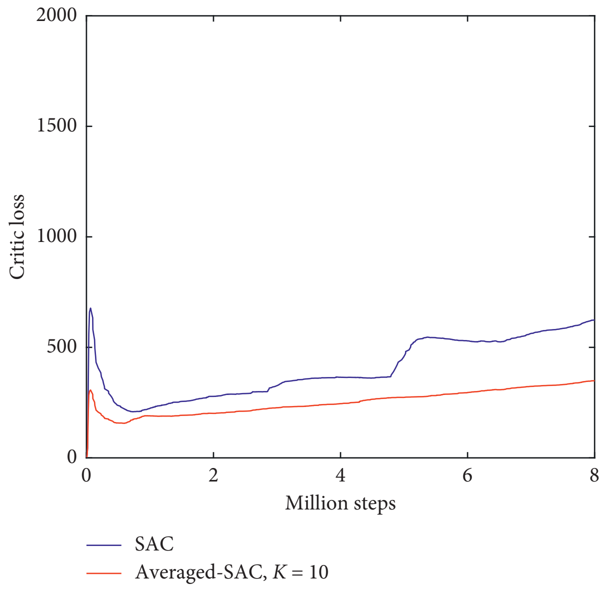


(b)

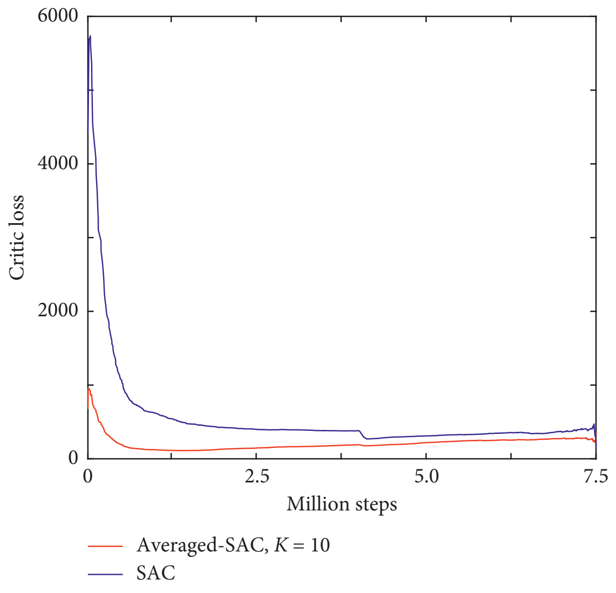
Figure 3: Continued.



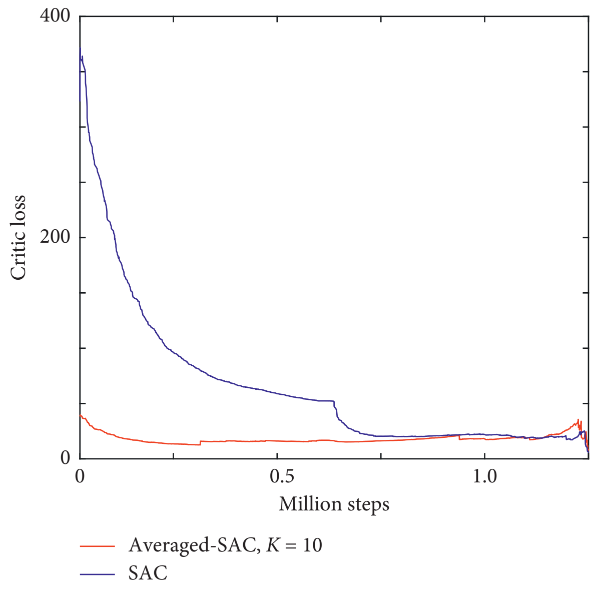
(c)



(d)



(e)

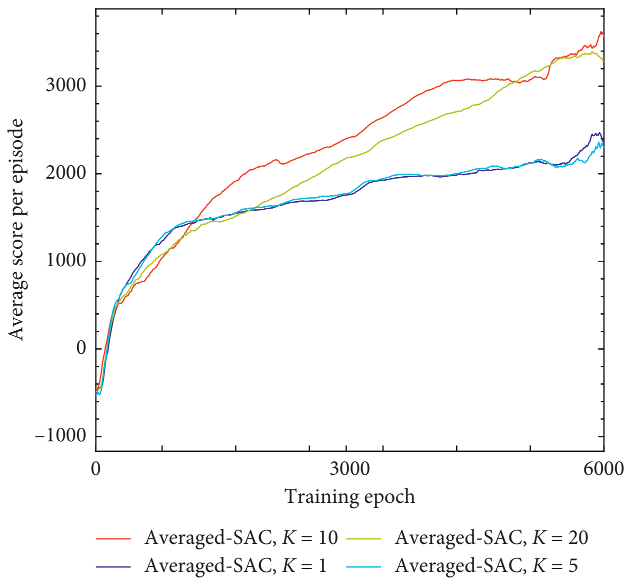


(f)

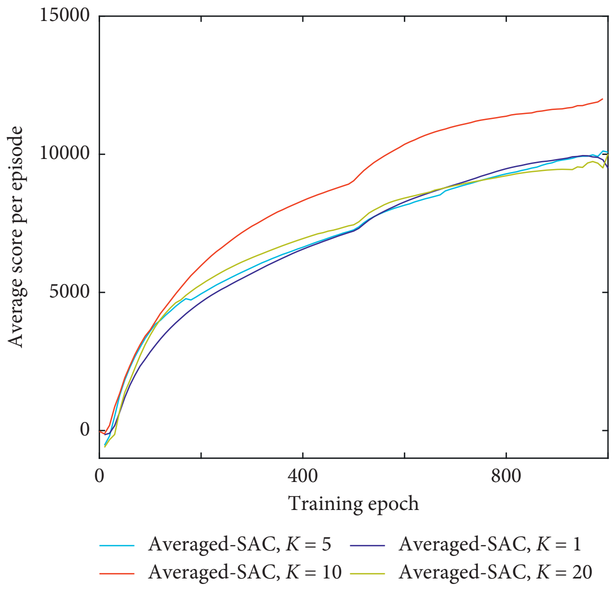
FIGURE 3: Results of the critic loss of Averaged-SAC and SAC in 6 MuJoCo games. (a) Ant-v2. (b) HalfCheetah-v2. (c) Hopper-v2. (d) Humanoid-v2. (e) Walker2D-v2. (f) InvertedDoublePendulum-v2.

addition, we concluded that as the value of K increases a little, the average training score of Averaged-SAC will increase; if the increased K value exceeds a certain limit, the average score of Averaged-SAC will show a decreasing trend in certain game environments. Since Averaged-SAC needs to calculate the average value, it needs more training time. But considering its performance improvement, the cost is acceptable.

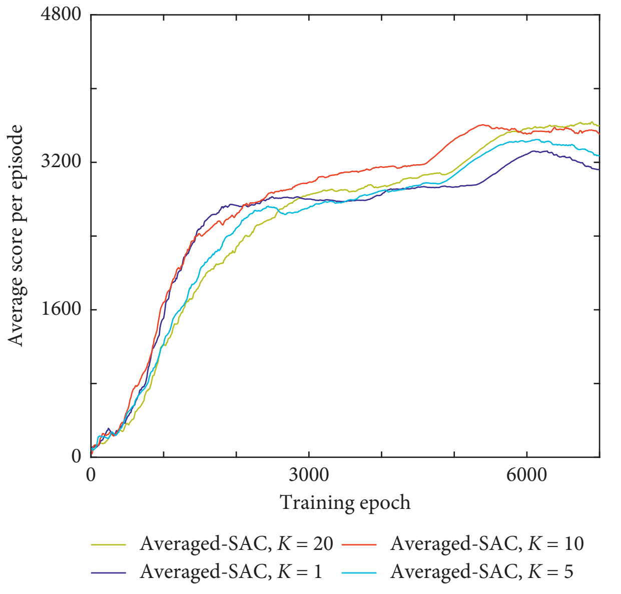
In summary, after 6 comparative experiments in the MuJoCo environment, we confirm that Averaged-SAC can indeed achieve better performance than SAC in MuJoCo games. In addition, increasing the value of K in Averaged-SAC appropriately will result in better performance and stability with acceptable increased training time costs. However, in Averaged-SAC, an appropriate value of K should be selected according to computing resources



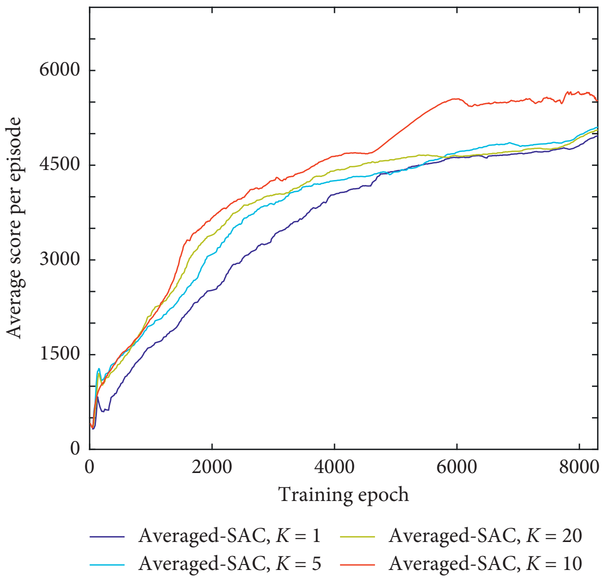
(a)



(b)

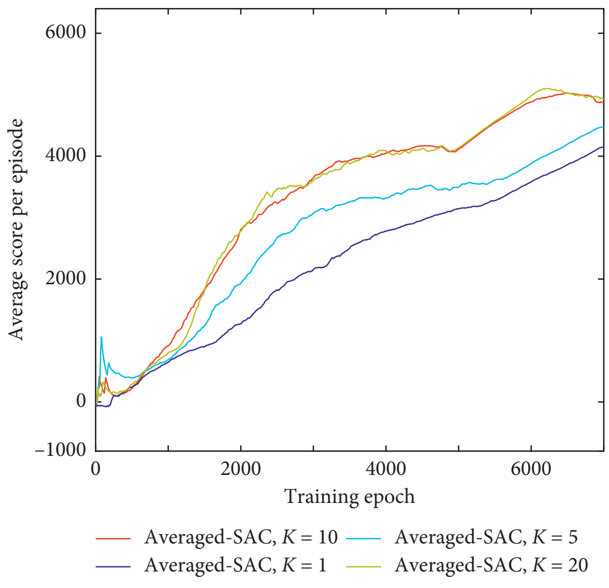


(c)

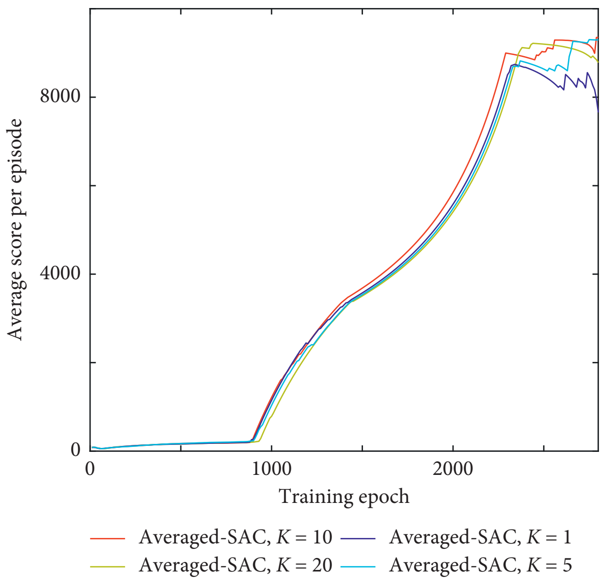


(d)

FIGURE 4: Continued.

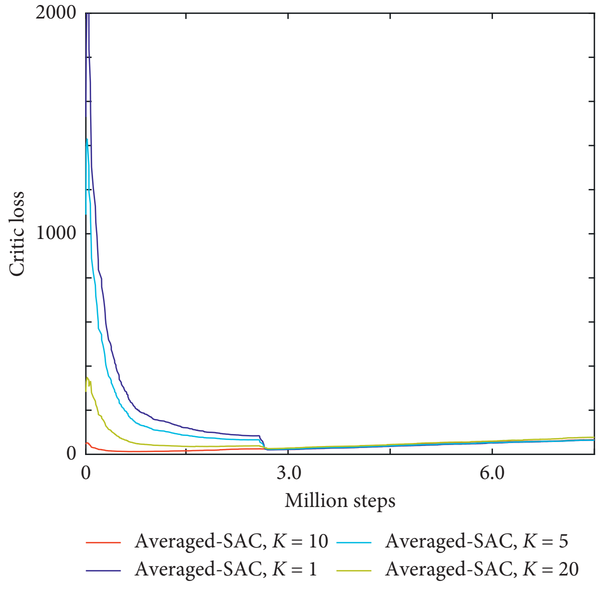


(e)

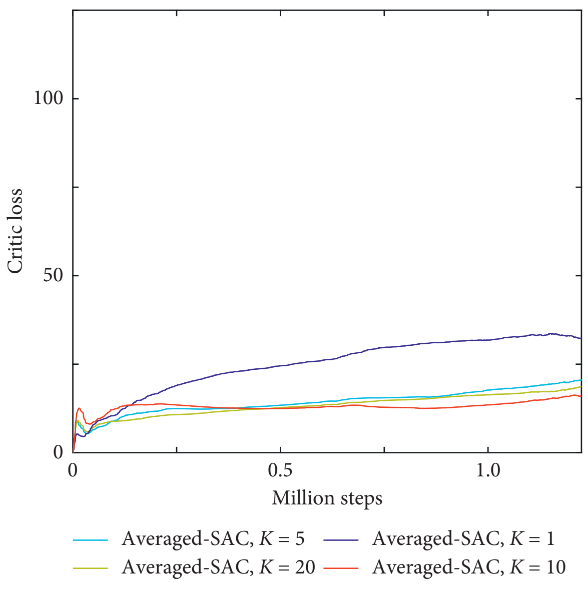


(f)

FIGURE 4: Results of the Averaged-SAC with different K values. (a) Ant-v2. (b) HalfCheetah-v2. (c) Hopper-v2. (d) Humanoid-v2. (e) Walker2D-v2. (f) InvertedDoublePendulum-v2.

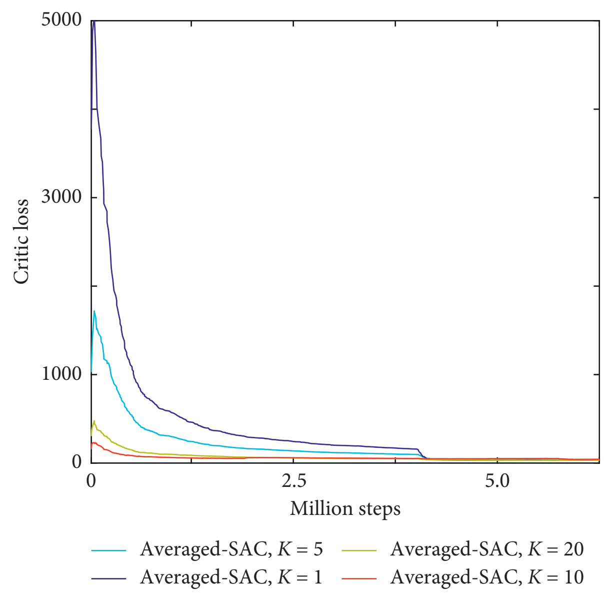


(a)

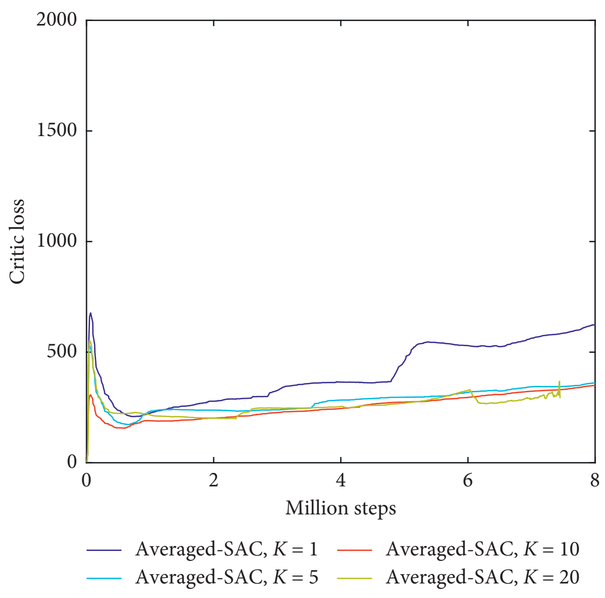


(b)

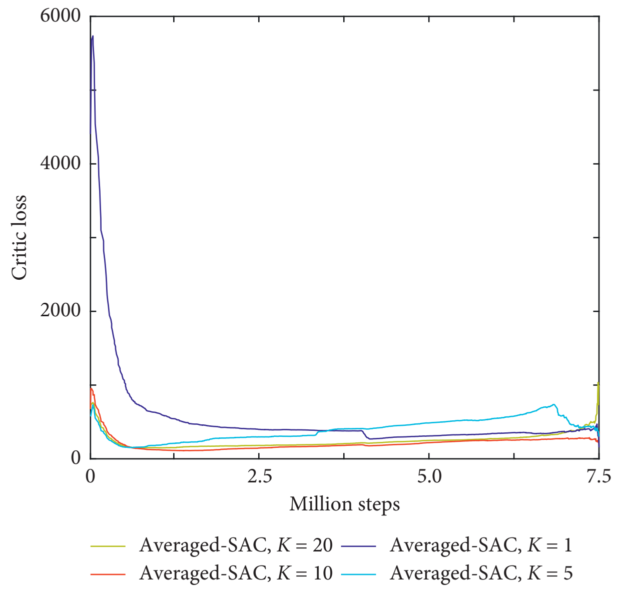
FIGURE 5: Continued.



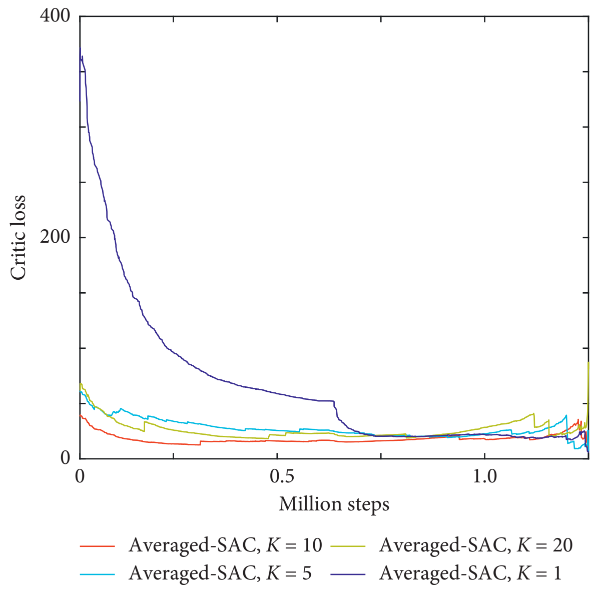
(c)



(d)



(e)



(f)

Figure 5: Results of the critic loss of Averaged-SAC with different K values. Analysis of the experimental performance of Averaged-SAC and SAC. (a) Ant-v2. (b) HalfCheetah-v2. (c) Hopper-v2. (d) Humanoid-v2. (e) Walker2D-v2. (f) InvertedDoublePendulum-v2.

Table 1: The training scores of Averaged-SAC and SAC in 6 MuJoCo games.

Environment	SAC	Averaged-SAC, $K = 10$
Ant-v2	2235.1 ± 260.5	2447.2 ± 180.3
Walker2D-v2	2725.4 ± 512.5	3027.6 ± 632.2
Hopper-v2	2322.7 ± 425.6	2468.4 ± 354.3
Humanoid-v2	3341.1 ± 547.9	3611.1 ± 637.9
HalfCheetah-v2	8813.2 ± 382.6	9613.8 ± 252.4
InvertedDoublePendulum-v2	8503.2 ± 122.6	9023.2 ± 112.3

because a larger value of K will require more training time costs.

6. Conclusion

This paper proposes an Averaged-SAC algorithm, which uses K previously learned state values to calculate the soft Q-value in the Averaged-SAC algorithm. The purpose is to effectively reduce the error caused by overestimation of soft Q-value in the calculation process. We have proved that Averaged-SAC can indeed stabilize training and improve performance in several games in MuJoCo's continuous action space. In addition, we also conducted some experiments to study the influence of K in the Averaged-SAC algorithm. Experimental results show that within a certain range, increasing the average number of previously learned state values will lead to better performance; if the increased average number of previously learned state values exceeds a range, the learning performance will show a decreasing trend; by repeatedly taking experiments with different K values, we find that when the K value is 10, the learning performance can reach the best. Averaged-SAC is a simple extension that can be easily integrated with SAC.

In future work, we plan to test Averaged-SAC on other off-policy DRL algorithms and other benchmarks (such as Atari games) to see if the significant performance improvement observed on MuJoCo can be extended to other algorithms and environments. In addition, investigations on the selection of K will be conducted. This means that we can dynamically understand how many previously learned soft Q-values should be used for averaging to achieve the best performance. A simple suggestion might be to associate the number of K with the state TD-error. Another method is to use neural networks to make K a dynamically learnable parameter. Finally, incorporating averaging techniques into policy gradient-based methods (such as PPO and TRPO methods) can further improve the performance of these algorithms.

Data Availability

We perform experiments on six MuJoCo games. The MuJoCo games used are commonly used public environments, which are linked below. MuJoCo: http://www.mujoco.org.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported in part by the National Natural Science Foundation of China under grant nos. 61672123, 61602083, and 62002044, the Fundamental Research Funds for the Central Universities under grant no. DUT20LAB136, and the China Scholarship Council.

References

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *Institute of Electrical and Electronics Engineers Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[2] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," 2013, https://arxiv.org/abs/1312.5602.

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

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

[5] H. V. Hasselt, "Double *Q*-learning," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 2613–2621, Vancouver, CA, USA, December 2010.

[6] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the International conference on machine learning (ICML)*, pp. 1–9, Atlanta, GA, USA, June 2013.

[7] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2017, https://arxiv.org/abs/1709.06560.

[8] B. Mazoure, T. Doan, A. Durand, R. D. Hjelm, and J. Pineau, "Leveraging exploration in off-policy algorithms via normalizing flows," 2019, https://arxiv.org/abs/1905.06893.

[9] W. Dabney, Z. Kurth-Nelson, N. Uchida et al., "A distributional code for value in dopamine-based reinforcement learning," *Nature*, vol. 2020, p. 5, 2020.

[10] J. Duan, S. E. Li, Y. Guan, Q. Sun, and B. Cheng, "Hierarchical reinforcement learning for self-driving decisionmaking without reliance on labeled driving data," *IET Intelligent Transport Systems*, vol. 14, 2020.

[11] O. Anschel, N. Baram, and N. Shimkin, "Averaged-DQN: variance reduction and stabilization for deep reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, pp. 176–185, Sydney, Australia, August 2017.

[12] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 2944–2952, Montreal, CA, USA, December 2015.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[14] D. Kingma and J. A. Ba, "A method for stochastic optimization," in *Proceedings of the International Conference for Learning Presentations (ICLR)*, San Diego, CA, USA, May 2015.

[15] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 1008–1014, Denver, CO, USA, June 2000.

[16] 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.

[17] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the In International Conference on Machine Learning (ICML)*, pp. 1889–1897, Lille, France, July 2015.

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

[19] P. Thomas, “Bias in natural actor-critic algorithms,” in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 441–448, Beijing, China, June 2014.

[20] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients,” *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[21] T. Haarnoja, A. Zhou, K. Hartikainen et al., “Soft actor-critic algorithms and applications,” 2018, https://arxiv.org/abs/1812.05905.