Deep Deterministic Policy Gradient With Compatible Critic Network

Di Wang, *Student Member, IEEE*, and Mengqi Hu, *Member, IEEE*

***Abstract*— Deep deterministic policy gradient (DDPG) is a powerful reinforcement learning algorithm for large-scale continuous controls. DDPG runs the back-propagation from the state-action value function to the actor network's parameters directly, which raises a big challenge for the compatibility of the critic network. This compatibility emphasizes that the policy evaluation is compatible with the policy improvement. As proved in deterministic policy gradient, the compatible function guarantees the convergence ability but restricts the form of the critic network tightly. The complexities and limitations of the compatible function impede its development in DDPG. This article introduces neural networks' similarity indices with gradients to measure the compatibility concretely. Represented as kernel matrices, we consider the actor network's and the critic network's training dataset, trained parameters, and gradients. With the sketching trick, the calculation time of the similarity index decreases hugely. The centered kernel alignment index and the normalized Bures similarity index provide us with consistent compatibility scores empirically. Moreover, we demonstrate the necessity of the compatible critic network in DDPG from three aspects: 1) analyzing the policy improvement/evaluation steps; 2) conducting the theoretic analysis; and 3) showing the experimental results. Following our research, we remodel the compatible function with an energy function model, enabling it suitable to the sizeable state-action space problem. The critic network has higher compatibility scores and better performance by introducing the policy change information into the critic-network optimization process. Besides, based on our experiment observations, we propose a light-computation overestimation solution. To prove our algorithm's performance and validate the compatibility of the critic network, we compare our algorithm with six state-of-the-art algorithms using seven PyBullet robotics environments.**

***Index Terms*— Compatible function, deep deterministic policy gradient (DDPG), overestimation, reinforcement learning (RL).**

I. Introduction

REINFORCEMENT learning (RL) discovers the best policy through repeatedly interacting with environments and maximizing the cumulative rewards. Building on the Markov decision process (MDP), states, actions, rewards, and the transition function are four essential parts. States represent the environment. The transition function records the probability when one state is transitioned to another. Moreover, based on the availability of the initial actions, the expectation of the return can be modeled as state value function or state-action value function (Q-value). The Q-value is the foundation of the deep deterministic policy gradient (DDPG) [1] method. With the consideration of the future rewards, Q-value measures the quality of an action at a given state followed by the current policy. To decrease the computation complexity, the temporal differences (TDs) trick [2] is applied. Formally, policy refers to the probability distribution over actions at each state. However, deterministic policy denotes the action to take from the state. With the deterministic policy, we can apply the chain rule to the Q-value. Thus, once the Q value is increased, the policy will be improved. Taking deep neural networks (DNNs) as function approximators, deep RL can solve large-scale problems and have achieved great success in various domains, including robot control [3]–[5], computer game [6], [7], economy prediction [8], and scheduling problems [9].

Deterministic policy lies behind the success of DDPG. However, this unique feature raises a critical challenge to the compatibility of the critic network. For example, advantage actor-critic (A2C) [10], with the likelihood-maximization process, modifies the policy distribution based on the current policy's advantage value. Small changes in the advantage value will not change the shape of the policy distribution hugely. In DDPG, any error in the state-action value function will influence the policy enormously via the back-propagation process. The noisy state-action value function is inevitable with the function approximator, experience replay buffer trick [11], and mini-batch trick. The performance of the DDPG depends on the cooperation between the actor network and the critic network [12], [13]. A critic network with poor compatibility will provide wrong gradient information to the actor network.

The compatibility issue comes from the work of Silver *et al.* [14], which makes significant contributions to the development of deterministic policy gradient (DPG). In their convergence proof, it is necessary to use a linear compatible function as the state-action value function (the critic part), which extracts the policy change information. The original loss function is the mean square error between the gradient of the estimated Q value over action and the real Q value over action. We usually substitute the original loss function with the mean square error between the estimated Q-value and the actual Q-value because of the difficulty of calculating these gradients. In DDPG, Lillicrap *et al.* [1] take neural networks

Manuscript received 18 December 2020; revised 21 July 2021; accepted 26 September 2021. Date of publication 15 October 2021; date of current version 4 August 2023. *(Corresponding author: Mengqi Hu.)*

The authors are with the Department of Mechanical and Industrial Engineering, University of Illinois at Chicago, Chicago, IL 60607 USA (e-mail: dwang66@uic.edu; mengqihu8@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3117790.

Digital Object Identifier 10.1109/TNNLS.2021.3117790

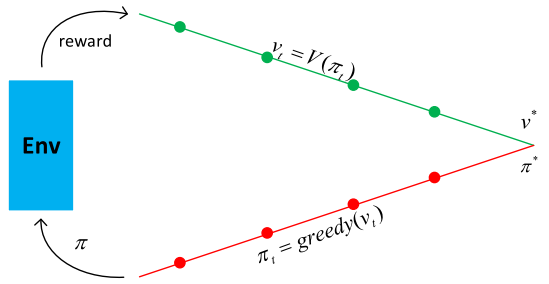


Fig. 1. Policy improvement and policy estimation steps in RL. It can converge to the optimal policy.

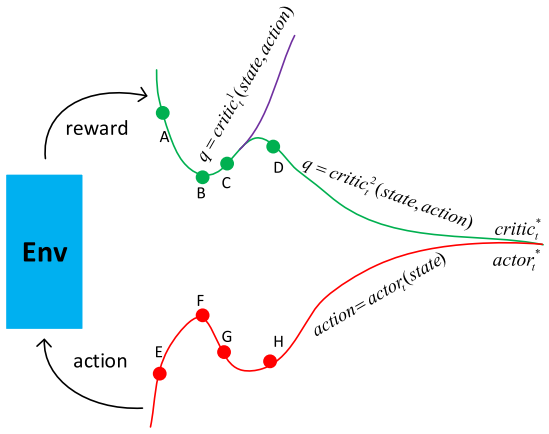


Fig. 2. Policy improvement and policy estimation in DDPG. With the offline setting, function approximation, noise added policy, and other tricks, the policy improvement and the policy evaluation steps become complex, which raises more challenges to critic network's compatibility. DDPG runs the back-propagation from the state-action value function to the parameters of the actor network, which intensifies the need for a compatible critic network.

as the function approximator to solve large state-action space problems, which drops the linear compatible function and the policy change information. We will show more details and challenges in Section IV-A.

A concrete example can demonstrate the compatibility issue clearly. As the two most essential steps in the RL algorithm, the policy evaluation step iteratively estimates state value function v_π. The policy improvement step greedily generates better policies, as illustrated in Fig. 1. Without the function approximation error, it can converge to the optimal policy without the risk of trapping in the local optimal. As shown in Fig. 2, things are more complicated. First, DDPG utilizes the DNNs as the function approximator, of which the performance during the training process highly varies based on the training dataset. With the experience buffer, the buffered dataset is highly biased, which is the cutting point of the prioritized experience replay trick [15]. Moreover, with the mini-batch trick, the training process of the actor network and the critic network are complicated and correlated. The success of the asynchronous advantage actor-critic (A3C) algorithm [16] proves that the reduction in correlation can improve the performance.

In DDPG, the actor network focuses on policy improvement, and the critic network aims for policy estimation. The parameters of the critic network are trained based on the sampled history data from the replay buffer. If the trained critic network is not compatible with the current policy, the performance will be poor. In Section IV-A, we will present detailed theoretic analysis. As shown in Fig. 2, the policy evaluation process and the policy improvement process highly depend on the neural network training process involving the sampled data and hyperparameter settings. From point E to point F, the actor network's performance improves. After training on the collected experience trajectory, the critic network keeps approaching a local-optimal solution. When the critic network traps into the local optimum, it will approximate the state-action value function as critic_t^1, which will mislead the actor network through the back-propagation process reaching a different policy improvement process. This wrong instruction hinders the improvement of the policy. Because of the high correlation, the mistakes in the policy evaluation will mislead the policy improvement process, in turn, accumulate errors in the policy evaluation, like the overestimation issue. However, the overestimation issue occurs mainly because of the maximization operator, which will be discussed later. As shown in [17], [18], the premature convergence to the local-optimal issue is remarkable in the policy gradient-based RL methods.

We typically add randomness to the policy to boost the exploration ability, such as the epsilon greedy method or adding some noise. Assuming the actor network reaches H by adding some noise, one of its compatible state-action value functions should be critic_t^2. The critic network needs to change from the critic^1 to critic^2 quickly with the newly generated data. Because critic^1 to critic^2 have totally different estimation at point D. critic^1 is incompatible with the current policy H, and critic^1 misleads the actor network through the back-propagation. Thus, there is an urgent need to rebuild the compatible function and combine it with the critic network.

The actor network and the critic network should be compatible. In other words, they should learn how to find or how to score the best policy with the same convergence speed. Motivated by this idea, we introduce the similarity of neural networks with a gradient to measure compatibility. First, we take a kernel function on the learned feature vectors and gradient vectors over each layer to calculate two kernel matrices. After integrating them with the Hadamard product, we represent the actor network and the critic network with one kernel matrix. The sketch trick is applied to decrease the computation time. The centered kernel alignment (CKA) index [19] and the normalized Bures similarity (NBS) index [20] are selected to calculate the compatibility scores, which are proved to be consistent with each other and the performance. Moreover, we propose a compatible critic network enabling the critic network to receive feedback information from the actor network. To incorporate the policy change information in the compatible function, we build an energy function model to represent this information. After transforming this energy function with a partition function, we obtain a rate measuring the policy varying degrees on the sampled transition data compared to the data stored in the replay buffer. Then we adjust the critic's loss function according to this rate. Empirically, we show that our proposed method has higher

performance and similarity scores. The physical explanation of the compatible critic network is listed as follows.

1) When policy changes largely, the critic will become radical and update largely on the newly generated data to make a time-urgent response to the actor network.
2) When policy changes subtly, the critic will be cautious and decrease its updating steps on the newly generated data. Thus we can avoid over-fitting and mitigate the abnormal interrupts in Q-values.

Overestimation bias is induced when deep RL maximizes the noisy state value function or state-action value function. Besides, this bias accumulates with the TD learning trick [2]. The overestimation issue is tough to mitigate because it is difficult to estimate the upper bound of the Q value for each state-action pair. The state-action space is vast, and the current policy's data distribution is different from that distribution of the optimal one. Another observation in our experiment shows that the peak of the overestimated Q value is tremendously bigger than the corresponded discounted return and the maximum accessible discounted return of the environment. This maximum accessible discounted return of the environment will be received only when the optimal policy is obtained. Thus, using the maximum accessible discounted return to solve the overestimation issue will not impede us from approaching the optimal policy. This finding motivates us to propose a lightweight method to estimate the maximum possible discounted return of the environment to bound the target Q value with a Monte Carlo method. Besides, the compatible critic network assists in mitigating the overestimation through checking the policy change information. The needs for a compatible critic network can also be illustrated via the delayed updating trick's success. In the delayed updating trick, the critic network updates parameters more frequently to catch up with the policy's change. However, this trick treats each state-action pair equally, which may cause the over-fitting issue at the local optimal point. In Section IV-C, we demonstrate that the undesirable use of the delayed trick may hurt the performance, which is caused by the incompatibility of the critic network.

The organization of the remainder of this article is as follows: Section II presents related works involving state-of-the-art RL algorithms, compatible function, neural network's similarity, and overestimation. Section III illustrates the foundation of DDPG and the calculation of similarity indices. In Section IV, we first discuss the existence and seriousness of the compatibility issue. Then we illustrate our proposed compatible critic network and overestimation solution. In Section V, we compare our algorithm with state-of-the-art RL algorithms. Future studies are discussed in Section VI.

II. Related Work

As a hot topic in artificial intelligence, RL attracts many researchers' eyes. Silver *et al.* [14] prove that DPG outperforms the stochastic policy gradient in applications with high-dimensional action space. Kakade [21] introduced the natural policy gradient to improve the convergence property of the policy gradient. By restricting the updating amplitudes of parameters, the policy objective function is updated toward the steepest direction. Similar to the natural policy gradient algorithm, Schulman *et al.* [22] propose a trust region policy optimization (TRPO) algorithm which can guarantee a monotonic improvement in expected reward functions using several approximation strategies. Furthermore, Schulman *et al.* [23] proposed a proximal policy gradient (PPO) algorithm by simplifying the calculation of TRPO. Wu *et al.* [24] present actor-critic using Kronecker-factored trust region (ACKTR) algorithm by introducing the Kronecker-factored approximation to speed up the calculation of the natural gradient. Mnih *et al.* [10] adopted parallel actors to update the shared policy model, which can stabilize the learning process and improve data efficiency. Haarnoja *et al.* [25] proposed a soft actor-critic (SAC) algorithm by introducing the energy function to maximize the policy's entropy and received rewards. Barth-Maron *et al.* [26] substituted the deterministic critic network with a distributional critic network to exploit more information, known as D4PG.

Compatibility issue is challenging in DDPG. Sutton *et al.* [27] studied the relationship between the linear approximated state-action value function and the policy gradient. The compatibility is emphasized in their function approximation theorem. Sutton *et al.* [28] propose a series of gradient TD algorithm to correct the gradient provided by the linear approximated function. Silver *et al.* [14] presented the compatible function approximation theorem in their DPG work, which serves as the foundation of the DPG. With the DNN's help, RL, such as DDPG, starts to solve massive state-action space problems. DDPG [1] takes neural networks as a nonlinear approximated function instead of the linear compatible function format. Balduzzi and Ghifary [29] proposed a compatible value gradient method using a neural network to generate the gradient of the value function. This work demonstrates the effectiveness of considering the compatibility issue in DDPG. Peters *et al.* [30] introduced a compatible function approximator to approximate the natural gradient of the A2C method.

The research of neural networks' similarity catches eyes because of their wide applications like transfer learning and meta-learning. Achille *et al.* [31] proposed to use "probe network" and Fisher Information Matrix to calculate the similarities of different visual tasks. Kornblith *et al.* [32] examined the performance of several similarity indices and prove that CKA can reliably identify correspondences between representations from different trained models. Tang *et al.* [33] introduce gradient vectors into similarity calculation and prove its feasibility.

It is proven that the overestimation in the discrete domain is common and harmful to the performance of RL algorithms [34]. Fujimoto *et al.* [35] prove the existence of the overestimation issue in DDPG. Two critic networks are used in their proposed TD3 method, where the estimate of one critic is used as the upper bound of the other critic. The minimum assessment of these two critics is used as a less biased value estimator. Wu *et al.* [36] prove the underestimation issue in the TD3 method. They suggest using the weighted sum of three critic estimates as the final estimate. Researchers pay great attention in this field [37], [38] because

DDPG back-propagates the gradients from the state-action value function to the actor's parameters. To reduce overestimation errors, double Q-learning [34] employs a pair of critics. The actions obtaining the maximum Q-values from the first critic are selected. The second critic provides value estimations of the chosen actions. Zhang *et al.* [39] find that double Q-learning sometimes underestimates Q-values and thus propose a weighted dual Q-learning algorithm to address this issue. Anschel *et al.* [40] use K previous critic estimates to calculate the current target critic estimate for the sake of variance reduction. Lee *et al.* [41] propose a bias-corrected Q-learning algorithm where all pairs of state-action data are needed to compute the bias-corrected term. He *et al.* [42] incorporate multistep rewards into the training process of Q-learning to speed up reward propagation. The authors also take the upper bound and the lower bound of Q-values as constraints in the training objective function. The lower bound is derived from the real Q-value's bound, assuming that the trained Q-value is close to the real one. However, the calculation of bounds is expensive, which requires iterating all available reward combinations. Nachum *et al.* [43] employ a smoothed target Q-value [44] to train a Gaussian policy that can reduce the high variance. De Asis *et al.* [45] propose a new $Q(\sigma)$ multistep reward TD algorithm where σ controls the tradeoff between the bias and the variance. Al-Dabooni and Wunsch [46] take a dual critic-network framework to estimate the Q values of the next time step and N steps later [47], respectively. They use the average of these two estimations to compute the target Q value.

III. Background

A. Deep Deterministic Policy Gradient

RL optimizes the policy of the agent with a sequence of agent-environment interactions. Typically, it models the problem as a MDP which means that given the state space S and the action space A, the stationary transition function satisfies $p(s_{t+1} \mid s_1, a_1, \ldots, s_t, a_t) = p(s_{t+1} \mid s_t, a_t)$. The agent iteratively optimizes its policy π with the goal of maximizing the performance objective function $J(\pi) = \mathbf{E}[R_t^\gamma \mid \pi]$, where the return is denoted in the following equation:

$$R_t^\gamma = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i), \quad 0 < \gamma < 1. \tag{1}$$

The state-action value function $Q^\pi(s_t, a_t)$ is the expectation of the return given s_t, a_t, π and the discounted state distribution $\rho^\pi(s') := \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) ds$ as in (2). The discounted state distribution shows that the next state s' after taking action a at time t follows distribution ρ^π [14]. $p_1(s)$ refers to the initial state distribution [14]

$$Q^\pi(s_t, a_t) = \mathbf{E}_{s \sim \rho^\pi, a \sim \pi}\left[R_t^\gamma \mid s_t, a_t\right]. \tag{2}$$

According to the chain rule, (2) can be rewritten as follows:

$$Q^\pi(s_t, a_t) = \mathbf{E}_{s \sim \rho^\pi}\left(r(s_t, a_t) + \gamma \mathbf{E}_{a_{t+1} \sim \pi}\left[Q^\pi(s_{t+1}, a_{t+1})\right]\right). \tag{3}$$

With the deterministic policy μ, we can remove the expectation $\mathbf{E}_{a_{t+1} \sim \pi}$ as in the following equation:

$$Q^\mu(s_t, a_t) = \mathbf{E}_{s \sim \rho^\beta}\left[r(s_t, a_t) + \gamma\, Q^\mu(s_{t+1}, \mu(s_{t+1}))\right] \tag{4}$$

where β refers to the stochastic behavior policy in the off-policy setting and $\beta(a \mid s) \neq \pi(a \mid s)$ [14]. In the off-policy setting, we apply the experience replay buffer to store the trajectories. β refers to the stored "old" policy which is different from π. In DDPG, we use the noise added deterministic policy to interact with the environment generating the trajectories. The corresponding discounted state distribution is denoted as $\rho^\beta(s') := \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \beta) ds$.

Representing the actor with parameter θ and the critic with parameter ω in the function approximations, the performance objective, and its gradient can be written as

$$J_\beta(\mu_\theta) = \int_S \rho^\beta(s) Q^\mu(s, \mu_\theta(s)) ds \tag{5}$$

$$\nabla_\theta J_\beta(\mu_\theta) \approx \mathbf{E}_{s \sim \rho^\beta}\left[\nabla_\theta \mu_\theta(s)\, \nabla_a\, Q^\mu(s, a)\mid_{a=\mu_\theta(s)}\right]. \tag{6}$$

Here, we make an approximation in (6) which will be discussed in Section IV-A. To make the above (6) workable, we need to replace the true state-action value function $Q^\mu(s, a)$ with the estimated state-action value function $Q^\omega(s, a)$. Thus, the loss function of the critic network can be further approximated as in (7) and (8). This approximation will be discussed in Section IV-A

$$L(w) = \mathbf{E}_{s_t \sim \rho^\beta, a_t \sim \beta}\left[(y_t - Q^\omega(s_t, a_t \mid \omega))^2\right] \tag{7}$$

$$y_t = r(s_t, a_t) + \gamma\, Q^{\omega'}\left(s_{t+1}, \mu_{\theta'}(s_{t+1}) \mid \omega'\right) \tag{8}$$

where ω' and θ' denote the parameters for target critic network and actor network, respectively, and y_t is written in the temporal-difference learning form.

B. Similarity Index

According to the article [33], given the dataset $D_{n=1}^N$, the kernel matrix K_f^l of feature vectors f_D^l and the kernel matrix K_g^l of gradient vectors g_D^l can be represented as

$$K_f^l = \left(\Phi_f^l\right)^T \Phi_f^l \succeq 0, \quad K_g^l = \left(\Phi_g^l\right)^T \Phi_g^l \succeq 0 \tag{9}$$

where $\Phi_f^l = [f_1^l, f_2^l, \ldots, f_N^l] \in \mathbb{R}^{d^l \times N}$, and $\Phi_g^l = [g_1^l, g_2^l, \ldots, g_N^l] \in \mathbb{R}^{d^l \times N}$. D refers to the data in the experience replay buffer of DDPG. N is the size of buffer. l is the lth layer of the neural network. d^l is the dimension of the output of the lth layer. Practically, f_n^l is the output's feature vectors of the lth layer with the input of the nth data in the dataset. g_n^l is the output's gradient vectors of the lth layer after one backward computation with the input of the nth data in the dataset. Integrating K_f^l and K_g^l with the Hadamard product $\circ$, we can get the lth layer's kernel matrix $K^l = K_f^l \circ K_g^l$ of a given neural network.

With the linear kernel, CKA [19] and NBS [20] are invariant to the isotropic scaling and rotation in the embedding space. The CKA score $\mathrm{CKA}(\theta, \omega)$ between the actor network (parameterized as θ) and the critic network (parameterized as ω) is

defined as [33]

$$\mathrm{CKA}(\theta,\omega)=\sum_{l_\theta=1,l_\omega=1}^{l_\theta=L_\theta,l_\omega=L_\omega}\mathrm{cka}\left(K_\theta^{l_\theta},K_\omega^{l_\omega}\right) \tag{10}$$

$$\mathrm{cka}\left(K_\theta^{l_\theta},K_\omega^{l_\omega}\right)=\frac{\left\langle K_\theta^{l_\theta},K_\omega^{l_\omega}\right\rangle_F}{\left\|K_\theta^{l_\theta}\right\|_F\left\|K_\omega^{l_\omega}\right\|_F} \tag{11}$$

where F is the Frobenius norm. L_θ and L_ω are the number of layers in the actor network and the critic network. Similarly, the NBS score $\mathrm{NBS}(\theta,\omega)$ is defined as [33]

$$\mathrm{NBS}(\theta,\omega)=\sum_{l_\theta=1,l_\omega=1}^{l_\theta=L_\theta,l_\omega=L_\omega}\mathrm{nbs}\left(K_\theta^{l_\theta},K_\omega^{l_\omega}\right) \tag{12}$$

$$\mathrm{nbs}\left(K_\theta^{l_\theta},K_\omega^{l_\omega}\right)=\frac{\mathrm{Tr}\left(K_\theta^{l_\theta\,1/2}K_\omega^{l_\omega}K_\theta^{l_\theta\,1/2}\right)}{\mathrm{Tr}\left(K_\theta^{l_\theta}\right)^{1/2}\mathrm{Tr}\left(K_\omega^{l_\omega}\right)^{1/2}}. \tag{13}$$

Because the computation time of calculating similarity index is $\mathcal{O}(N^3)$, we use the sketch trick to decrease the calculation time to $\mathcal{O}(M^3)$ where $M \ll N$ and we use $M=256$ following [33]. Sketch trick allocates N data into M buckets with a random projection and replaces the original dataset with M buckets. Accuracy proof and details can be found in [33].

C. Overestimation Issue in DDPG

According to the article [35], overestimation bias exists in DDPG. The proof is based on the assumption as

$$\mathbf{E}_{s\sim\rho^\beta}\left[Q^\omega(s,\mu_{\bar\theta}(s))\right]=\mathbf{E}_{s\sim\rho^\beta}\left[Q^\mu(s,\mu_{\bar\theta}(s))\right] \tag{14}$$

where $\forall\bar\theta\in[\theta_0,\theta_0+\delta(\theta^*-\theta_0)],\delta>0$. θ_0 denotes the original policy and θ^* refers to the optimal policy. Equation (14) assumes that there exists a $\bar\theta$ such that the expectation of the parameterized state-action value function equals to the expectation of the true state-action value function. Then we can derive that $Q^\omega(s,\mu_{\bar\theta}(s))-Q^\omega(s,\mu_{\theta_0}(s))\geq Q^\mu(s,\mu_{\theta^*}(s))-Q^\mu(s,\mu_{\theta_0}(s))\geq Q^\mu(s,\mu_{\bar\theta}(s))-Q^\mu(s,\mu_{\theta_0}(s))$, which demonstrates the overestimation bias. Detailed proof can be found in the article [35].

IV. DDPG With Compatible Critic Network

A. Critic Network's Compatibility Issue

In this section, to better understand the critic network's compatibility issue in DDPG, we will go over each assumption and approximation made in DPG and DDPG. DPG is the theoretical foundation of DDPG. The difference between them lies in that DPG uses the linear function approximation to guarantee its convergence in theory. However, DDPG adopts the neural network as the nonlinear function approximation to solve the large state-action space problem. In (6), $\nabla_\theta Q^{\mu_\theta}(s,a)$ is ignored according to [48]

$$\nabla_\theta J_\beta(\mu_\theta)=\mathbf{E}_{s\sim\rho^\beta}\left[\nabla_\theta\mu_\theta(s)\,\nabla_a\,Q^\mu(s,a)+\pi(a\,|_s)\,\nabla_\theta\,Q^{\mu_\theta}(s,a)\right]. \tag{15}$$

More details can be found at the Off-Policy Policy-Gradient Theorem in Degris *et al.* [48]. Given $U\subset\mathbf{R}^{\mathbf{N}_\mu}$ a nonempty, compact set, we define two policy sets

$$Z=\left\{\mu\in U\mid\nabla_\theta\left(\int_S\rho^\beta(s)Q^\mu(s,\mu_\theta(s))ds\right)=0\right\} \tag{16}$$

$$\overline{Z}=\left\{\mu\in U\mid\int_S\rho^\beta(s)\,\nabla_\theta\,\mu_\theta(s)\,\nabla_a\,Q^\mu(s,a)ds=0\right\} \tag{17}$$

where Z is the true set of local maxima and $\overline{Z}$ is the set of local maxima calculated with the approximation. The most important conclusion is $\overline{Z}\subset Z$. Detailed proof can be found in article [48]. This conclusion demonstrates that the policy space becomes more perplexing with more local optimal points using this approximation. This approximation performs well, but this approximation raises more challenges when we consider the critic network's compatibility issue.

In (7) and (8), we first replace the true state-action value function $Q^\mu(s,a)$ with the approximate state-action value function $Q^\omega(s,a)$ parameterized by ω. This replacement introduces function approximation errors. According to the Compatible Function Approximation Theorem [14], Q^ω should satisfy extra conditions to maintain the policy gradient as follows.

1) The linear relationship exists between $\nabla_a Q^\omega(s,a)$ and $\nabla_\theta\mu_\theta(s)$, because we use the mean square error to train $Q^\omega(s,a)$ to replace $Q^\mu(s,a)$ [14]

$$\nabla_a Q^\omega(s,a)\,|_{a=\mu_\theta(s)}=\nabla_\theta\mu_\theta(s)^T\omega. \tag{18}$$

2) The loss function of ω is to minimize

$$\mathbf{MSE}(\theta,\omega)=\mathbf{E}\left[\mathrm{error}(s;\theta,\omega)^T\mathrm{error}(s;\theta,\omega)\right] \tag{19}$$

$$\mathrm{error}(s;\theta,\omega)=\nabla_a Q^\omega(s,a)\,|_{a=\mu_\theta(s)}-\nabla_a\,Q^\mu(s,a)\,|_{a=\mu_\theta(s)}\,. \tag{20}$$

Satisfying these two conditions, we can derive $\mathbf{E}[\nabla_\theta\mu_\theta(s)\,\nabla_a\,Q^\omega(s,a)\,|_{a=\mu_\theta(s)}]=\mathbf{E}[\nabla_\theta\mu_\theta(s)\,\nabla_a\,Q^\mu(s,a)\,|_{a=\mu_\theta(s)}]$ from $\nabla_\omega\mathbf{MSE}(\theta,\omega)=0$. Detailed proof can be found in article [14]. In DPG, to satisfy the first condition, the authors suggest to use a linear compatible function as follows:

$$Q^\omega(s,a)=(a-\mu_\theta(s))^T\,\nabla_\theta\,\mu_\theta(s)^T\omega+V^v(s) \tag{21}$$

where $V^v(s)$ denotes the baseline function. The linear function works fine for small state-action space. Besides, due to the difficulty of sampling $\nabla_a Q^\mu$, DPG uses the temporal-difference error to replace condition 2, which provides a good approximation. The neural network has a more powerful representation ability to solve the sizeable state-action space problem. As discussed in article [1], although the use of neural networks breaks the first condition and loses the theoretical performance guarantee, DDPG performs well with the target network trick. However, the unsatisfaction of both conditions in (18) and (19) makes the critic network neglecting the policy change information and then raises more challenges to the critic network's compatibility.

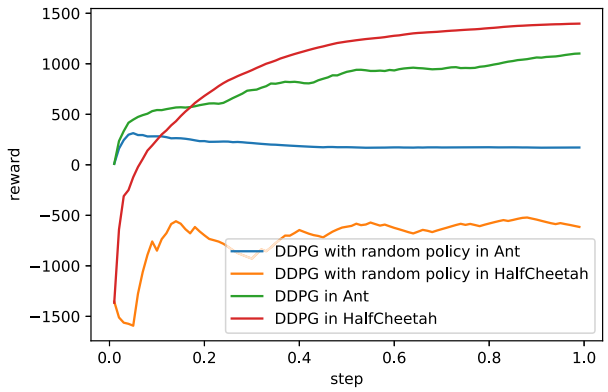


Fig. 3. Poor performance of DDPG with incompatible critic network. The figure is plotted with the testing data. In the testing, we use the actor network to generate actions. The random policy means that we only use randomly generated actions instead of the actor network's output during the training process. We can guarantee that the critic network is totally incompatible with the actor network with the random policy.

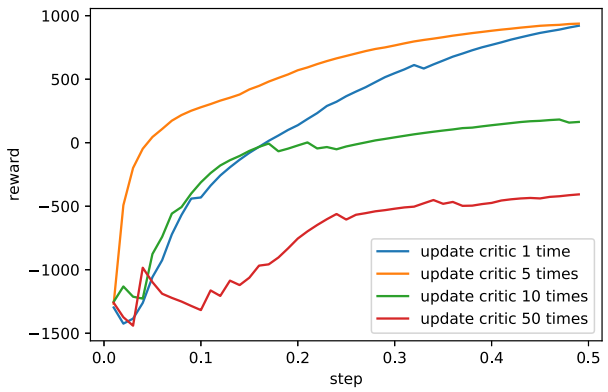


Fig. 4. Inadequate use of the delayed update trick ruins the performance. In the delayed update trick, the critic network unilaterally updates its parameters more often, causing over-fitting issues, decreasing the performance, and hindering compatibility. We need the policy change information in the optimization of the critic network.

Fig. 3 demonstrates the seriousness of the incompatible critic network. To test the compatibility, we use a random policy generator to interact with the environment. The random policy means that we only use randomly generated actions instead of the actor network's output during the training process. With the random policy, we can guarantee that the critic network is totally incompatible with the actor network. We test the performance of the actor network in different test environments. The figure is plotted with the testing data. The training steps are $1e^6$, and after each $1e^4$ time steps, we run ten times testing. In the testing, we use the actor network to generate actions. The HalfCheetah and Ant environments are tested in this figure. It is clear that the actor network instructed by a wholly incompatible critic network performs poorly. In the real case, we do not face this extreme condition, but we will have similar issues threatening the performance of the RL algorithms.

The delayed update trick updates the critic network more than once in each step to keep catching up with the new actor network, which seems similar to the compatible critic network. The difference lies in that the delayed update trick treats all sampled data the same without integrating the policy change information. Over-training on similar data will lead to over-fitting issues and make the critic network less sensitive to the newly generated data. In Fig. 4, we update the critic network one time, five times, ten times, and fifty times separately in the HalfCheetah environment to prove that inadequate use of the delayed update trick ruins the performance. The training time steps are $5e^5$, and after each $1e^4$ time step, we run ten times testing. The curve is plotted with the mean total rewards of ten runs of testing. The learning rate is 2^{-3}, which is different from other experiments. As the updating times of the critic networks increase at each time step, the performance drops hugely. Different sampled data are used at each critic training step, which makes it different from using a big learning rate. With the big learning rate, we update our neural networks' parameters aggressively based on the same sampled batch data. This experiment clearly shows that it is not enough for the critic to unilaterally update its parameters more often and treat each transition data equally. Besides, the delayed update trick cannot solve the compatibility issue. The delayed update trick performs well in practice, and we can solve the problem mentioned above by tuning the learning rate. However, here, we focus on the critic network's compatibility issue. This example is used to demonstrate the necessity of considering the policy changing information. Moreover, we test our proposed method empirically and find it can provide acceptable rewards with updating the critic network ten times at each time step or with a ten times bigger learning rate. Because they are not normal experiment settings, we will not put our results here.

B. Compatible Critic Network

Based on the above analysis and the compatible function's motivation, we propose a DDPG algorithm with a compatible function. Numerous successes prove the good performance of the temporal-difference error trick. We maintain the temporal-difference trick to provide the direction of the gradient $\nabla_\omega Q^\omega$. However, the magnitude of the gradient $\nabla_\omega Q^\omega$ is modified to integrate the policy change information. Our goal is to enable the critic network to respond to the changes in the policy quickly. Based on (21), we found that $|\nabla_\omega Q^\omega| \propto |a - \mu_\theta(s)|$ and $|\nabla_\omega Q^\omega| \propto |\nabla_\theta \mu_\theta(s)|$. We use the energy-based model to represent the policy change information as in (22). Similar to [25], [49], the general energy function can be denoted as $e^{(g(x))}$, where $g(x)$ is represented with a DNN

$$\varepsilon(s, a, \theta) = e^{(a-\mu_\theta(s))^2} + e^{\left(\mu_\theta(s)-\mu_{\theta'}(s)\right)^2}. \tag{22}$$

To eliminate the bad influence of abnormal values, we clip the values into a predefined range, which will be discussed later. Besides, because DDPG uses the soft target updates trick $\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tau \ll 1$, $\nabla_\theta \mu_\theta(s) = \mu_\theta(s) - \mu_{\theta'}(s)$ when $\theta - \theta'$ is small. With the experience replay buffer trick, the critic network is trained based on the replay buffer's sampled trajectories. Thus, the critic network mainly needs to respond to the newly added trajectories to the buffer. We take a baseline to form the partition function shown as (23), which

measures the performance of one trajectory compared with the whole trajectories in the replay buffer

$$\text{rate}_i = \text{clip}\left(\frac{\varepsilon(s_i, a_i, \theta)}{\frac{1}{N}\sum_{j=1}^{N}\varepsilon(s_j, a_j, \theta)}, 1-\lambda, 1+\lambda\right) \quad (23)$$

where λ is a hyperparameter to control the influence of abnormal values. The subscript i refers to the ith data point in the replay buffer. Typically, randomness will be added to the policy to boost the exploration ability. $a - \mu_\theta(s)$ will abnormal large with noisy. Empirically, we select a large value $\lambda = 0.7$. Its sensitive analysis and recommend value are shown in Section IV-D. N is the size of the replay buffer. The new loss function for the critic network can be rewritten as in (24). In practice, it is not necessary to update the baseline at each time step. Thus we set another hyperparameter f to control the updating frequency of the baseline

$$L(\omega) = \frac{1}{N}\sum_{i=1}^{N}\text{rate}_i \cdot (y_i - Q^\omega(s_i, a_i))^2. \quad (24)$$

Thus, the updating step length will increase with a responding large rate, and decrease with a small rate according to (23). When our algorithm converges, $a - \mu_\theta(s) = 0$ and $\mu_\theta(s) - \mu'_\theta(s) = 0$, then $\varepsilon(s, a, \theta) = 2$. We can get $\text{rate}_i = 1$ which degrades (24) to (7).

This property guarantees the performance of our algorithm is no worse than the performance of DDPG. Similar to the proof process in the compatible function approximation theorem in [14], we use temporal-difference learning to guarantee $Q^\omega(s, a) \approx Q^\mu(s, a)$, which will approximately satisfy the second condition in (19). By proving the satisfaction of these two compatible function approximation conditions, we can guarantee our algorithm following the true gradient, according to [14]. Besides, based on Sutton's work [28], we know that the temporal-difference learning method with the true gradient will converge. Thus, we can prove the convergence property of our proposed method.

The updating operation exists because of the calculation of $(1/N)\sum_{j=1}^{N}\varepsilon(s_j, a_j, \theta)$ in (23), which requires the updated actor network inputs all the data in the replay buffer. The computational complexity of this operation is high even we only use the data in the replay buffer instead of the whole dataset. Koenig and Simmons [50] proved that the complexity of the Q learning is the exponential function of the state space, which is intractable in the environment with large-scale and continuous state-action spaces. To have a brief view, we analyze the time complexity of our proposed method simply and compare it with other methods. Assume the length of each episode is d, and we need m episodes to train the neural networks. Thus, the time complexity for A2C and PPO is $d \times m$ because A2C and PPO are online deep RL methods. They typically visit each data point once. The time complexity for DDPG, SAC, D4PG, and TD3 are similar. With the replay buffer, they visit each data point multiple times. Assume each data point will be sampled s times at each time step. Thus, their complexity will be $m^2 \times d^2 \times s$. Assume the size of the replay buffer is b. For our method, with the updating operation, the complexity will be $m^2 \times d^2 \times s + m \times d \times b$. There are

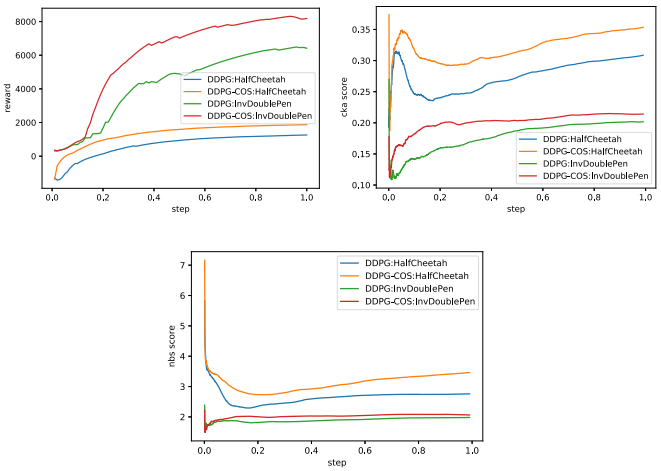


Fig. 5. Our proposed algorithm DDPG-COS surpasses DDPG in reward, CKA similarity score, and NBS similarity score. The performance, NBS index and CKA index are consistent, which proves the compatible critic network can improve the compatibility and the performance.

two methods to decrease the time complexity further. The first one is to set the updating frequency, which means that we only calculate $(1/N)\sum_{j=1}^{N}\varepsilon(s_j, a_j, \theta)$ after each f time steps. f denotes the frequency. Thus, the time complexity will be declined to $m^2 \times d^2 \times s + m \times d \times b/f$. In this article, we use this method. The second one is to apply the sketching trick. The dataset in the reply buffer will be allocated into z buckets with a random project, like a scaled matrix with Gaussian random variables. The size of each bucket is k_i, $i = 1, 2, \ldots, z$. Then the replay buffer will be replaced by the buckets in the calculation of $(1/N)\sum_{j=1}^{N}\varepsilon(s_j, a_j, \theta)$. Thus the time complexity will be $m^2 \times d^2 \times s + m \times d \times k_i$.

Fig. 5 is plotted with the testing data. The training steps are $1e^6$, and after each $1e^4$ time steps we run ten times testing and calculate the similarity indices. CKA and NBS scores are high at the initial time step because the actor network and the critic network have similar neural network frameworks, initial parameters, and a similar gradient direction. In the first $2e^5$ time steps of the HalfCheetah environment, the CKA score increases sharply with the rapid growth of the received rewards. When the increasing rate of the received rewards slows down, the CKA score and the NBS score drop slightly and then continue to increase. These peaks and valleys prove that compatible challenges exist during the training process. Especially when the policy changes fast, the compatibility issue should be paid more attention. Similar phenomena can be found in the InvDoublePen environment. Besides, it is shown that the range of the similarity scores are different in different environments because each environment has its unique task and action space. Moreover, our proposed method surpasses DDPG in performance and compatibility with higher rewards and similarity scores. In particular, in the HalfCheetah environment, DDPG with compatible critic network and overestimation solution (DDPG-COS) has a lower declining rate and higher increasing rate in both similarity scores.

C. Overestimation Solution

The overestimation issue happens because of $Q^\omega(s, \mu_\theta(s)) > Q^\mu(s, \mu_{\theta^*}(s))$. As denoted in Section III,

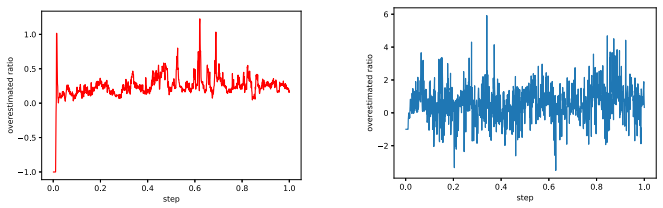


Fig. 6. (Left) Average overestimated Q value surpasses the average discounted return largely in DDPG with the InvertedDoublePendulum environment. The average overestimated ratio denotes the difference between the average estimated Q value and the average discounted return divided by the average discounted return. (Right) Overestimation ratio on a certain state-action pair during the training process.

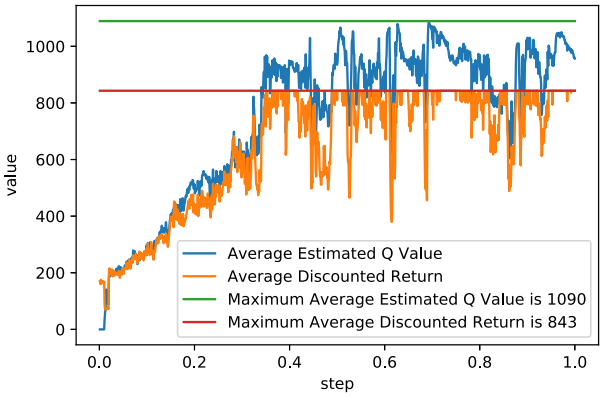


Fig. 7. Average estimated Q value and average discounted return of DDPG in the InvertedDoublePendulum environment. 52.1% of the collected average estimated Q values are larger than the maximum average discounted return, and the peak of the average estimated Q values is 29% larger than the maximum average discounted return.

θ denotes the parameterized actor network and ω is the parameterized critic network. μ refers to the deterministic policy. The difficulty of estimating $Q^{\mu}(s, \mu_{\theta^*}(s))$ lies in the toughness of sampling $\mu_{\theta^*}(s)$ and the large size of the state-action space. The Monte Carlo method samples data at the end of the trajectory, which provides a less biased state-action value function than the bootstrapping method. However, stochastisity exists at each time step of the trajectory. The state-action value function estimated by the Monte Carlo method has a higher variance by accumulating the stochastic in the trajectory. In our experiment, we find that the overestimated action value $Q^{\omega}(s, \mu_{\theta})$ largely surpass the corresponded true action value $Q^{\mu}(s, \mu_{\theta^*}(s))$ and the maximum accessible action value of the environment $Q^{\mu}_{\max}$.

Fig. 6 (left) plots the overestimation ratio, which is calculated by using the average estimated Q value minus the average discounted return and then dividing by the discounted return. According to [35], to calculate the average discounted return, we first sample 1000 states from the replay buffer as the starting states, then run these 1000 episodes following the current policy and average all collected discounted rewards over each state-action pair. For the discounted reward calculation in environments with infinite time steps, we only consider 1000 time steps because $0.99^{1000} = 4.31e^{-5}$ and the maximum time steps of the Pybullet environments is 1000 mostly. The overestimated ratio of a certain state-action pair cannot measure the overestimation issue clearly. The true Q value of each state-action pair is intractable. The training time steps in the InvertedDoublePendulum environment are $1e^6$. After each $1e^4$ step, we calculate the overestimation ratio. Moreover, the maximum overestimated ratio is 1.23. Besides, 24.3% of the testing data have a more than 30% overestimated ratio. Fig. 6 (right) plots the overestimation ratio on a certain state-action pair during the training process, where the overestimation issue is worse. Besides, we can also find the underestimation issue, which is not the focus of our article. Fig. 7 illustrates the average estimated Q value and the average discounted return. 52.1% of the collected average estimated Q values are larger than the maximum average discounted return, and the peak of the average estimated Q values is 29% larger than the maximum average discounted return.

This finding motivates us to use a light-computation method to estimate $Q^{\mu}_{\max}$ and use $Q^{\mu}_{\max}$ to bound the y value in (8). The advantages of this simplification include.

1) The computation is tractable. We decrease the complexity of the problem hugely. The large estimation space shrinks to one single point.
2) When we use the Monte Carlo method to estimate $Q^{\mu}_{\max}$, the variance decreases because of the maximizing operator.
3) The $Q^{\mu}_{\max}$ can be obtained when the best policy is adopted. Our goal is to find the best policy in accord with $Q^{\mu}_{\max}$. And it is harmful when the estimated Q value on any given state-action pair is much higher than the maximum Q value that existed. Our method can deal with this condition correctly.

However, when the overestimated value is less than $Q^{\mu}_{\max}$, this method will not provide any improvement. $Q^{\mu}_{\max}$ can be obtained with the following equation:

$$Q^{\mu}_{\max} = \max_{0 \le k \le K, 0 \le t \le T_k} R^{\gamma}_{k,t} \tag{25}$$

where T_k denotes the length of the kth trajectory and K denotes the number of trajectories. The loss function for the critic network can be rewritten as

$$L(w) = \mathbf{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta}\left[(y_t - Q^{\omega}(s_t, a_t \mid \omega))^2\right] \tag{26}$$

$$y_t = \text{clip}\Big(r(s_t, a_t) + \gamma\, Q^{\omega'}\left(s_{t+1}, \mu_{\theta'}(s_{t+1}) \mid \omega'\right) -\infty, Q^{\mu}_{\max}\Big). \tag{27}$$

We apply the clip operation here to directly limit the range of the target Q value. Based on our previous analysis, we find the estimated Q value on state-action pairs usually surpass $Q^{\mu}_{\max}$ largely. When the overestimated bias accumulates through the temporal-difference learning, the overestimation issue will be worse. Thus we limit the accumulation of overestimated bias directly by clipping the target Q value.

Compare to Fig. 7, we plot the average estimated Q value and the average discounted return of DDPG with our proposed overestimation solution. We use the same way to generate data. The replay buffer used in this article is $5e^4$, which is large enough to support a good estimate of the initial $Q^{\mu}_{\max}$ value. In Fig. 8, 18.4% of the average estimated Q values are larger

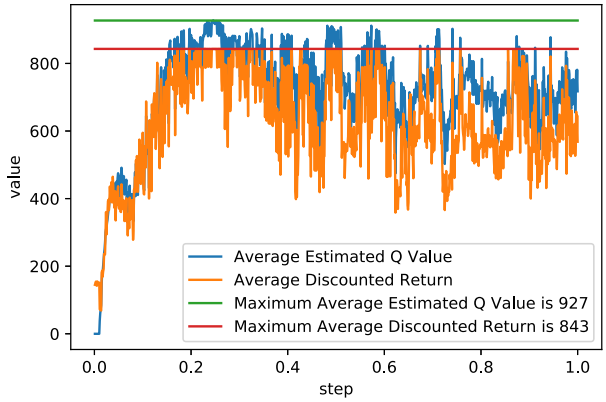


Fig. 8. Average estimated Q value and average discounted return of DDPG with our proposed overestimation solution in the InvertedDoublePendulum environment. 18.4% of the average estimated Q values are larger than the maximum average discounted return. And the peak of the average estimated Q value is 9% larger than the maximum average discounted return.

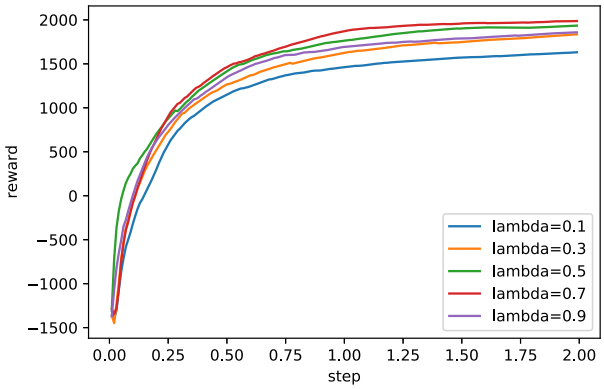


Fig. 9. Sensitivity analysis of λ value in HalfCheetah environment.

Algorithm 1 Pseudocode of the Proposed Algorithm

Initialize critic networks $Q_\omega(s, a \mid \omega)$, target critic networks $Q_{\omega'}(s, a \mid \omega')$ with $\omega' \leftarrow \omega$, actor network $\mu_\theta(s)$, target actor network $\mu_{\theta'}(s)$ with $\theta' \leftarrow \theta$, experience reply buffer with size N, hyperparameter λ, baseline updating frequency f, initial $Q^{\mu}_{\max}$
for episode $k = 1$ to K **do**
  for $t = 1$ to T_k **do**
    Observe state s_t, and generate action $a_t = \mu_\theta(s_t) + \eta_t$ where η_t denotes a random process
    Execute action a_t, receive reward r_t, next state s_{t+1}
    If environment is done, update $Q^{\mu}_{\max}$ with Eq. (25)
    Add one transition (s_t, a_t, r_t, s_{t+1}) to reply buffer
    Sample a minibatch of M transitions
    Calculate y value based on Eq. (27)
    Update critic with the loss function in Eq. (24)
    Update actor with the performance function in Eq. (6)
    Update the target actor and the target critic networks with the soft update trick
    Check the baseline updating frequency f. If the baseline updating time comes, update the baseline with Eq. (23)
  end for
end for

than the maximum average discounted return. And the peak of the average estimated Q value is 9% larger than the maximum average discounted return. We have mitigated the overestimation issue tremendously.

D. Algorithm

The pseudocode of the proposed algorithm is shown in Algorithm 1. Here, the hyperparameter λ is used to restrict the range of the parameter rate_i in (23). Because rate_i measures a ratio by comparing the energy-function changes (representing the policy changes) of the sampled transition with the energy-function changes of all transitions in the replay buffer, λ should be in the range of $[0, 1]$. We will analyze the boundary conditions here to present the influence of the λ value. If $\lambda = 0$, the information about the policy changes will not be counted. If $\lambda = 1$ and $\text{rate}_i = 0$, the sampled transition will not be considered again to avoid over-fitting since our model has been trained well on this transition data. If $\lambda = 1$ and $\text{rate}_i = 2$, we will increase the critic's updating step to two times. Because the critic network needs to modify its estimate largely to catch up with the current policy. From the viewpoint of the data sampling, the differences of our method with the prioritized experience replay trick [15] are summarized as follows.

1) Prioritized experience replay trick performs well by modifying the weights of each transition based on the temporal-difference error. Compared with our method, we introduce the policy change information to keep the critic network compatible. Our goal is different from the prioritized experience replay trick but preserves the additional benefits of treating each transition differently and paying attention to the difference among transitions. Our method can also treat the Blind Cliffwalk example well since our approach focuses on policy changes. This example is the motivating example of the prioritized experience replay trick. Details can be found in the article [15].
2) Prioritized experience replay trick changes the uniform sampling, introducing problems like bias and data diversity. The prioritized experience replay trick uses important-sample weights and rank-based prioritization to solve the issues mentioned above. Our approach keeps Uniform Sampling.

Fig. 9 demonstrates the mean total rewards received of 10 runs of testing in the HalfCheetah environment at each $1e^4$ time steps. The training steps are $2e^6$. Details of other parameters are described in Section V. We test four different λ values, and the best received mean rewards are 1631.51, 1835.27, 1934.43, 1986.60, and 1857.54, respectively. As is shown, the performance keeps increasing as the λ value rises. This phenomenon demonstrates the necessity of considering the compatibility issue during our training process. According to Fig. 9, we select $\lambda = 0.7$ in this article.

In Fig. 10, we recalculate the value of the baseline in (23), after each f time steps, to further decrease the computation.

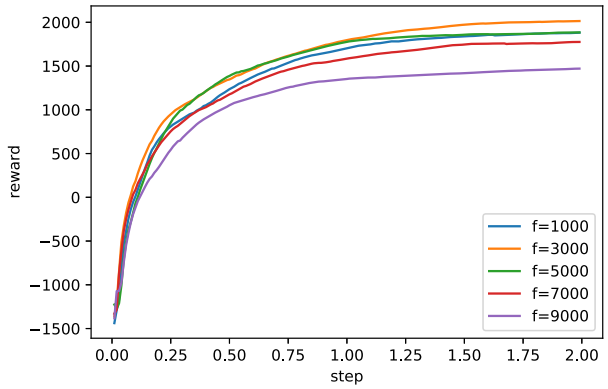


Fig. 10. Sensitivity analysis of f value in HalfCheetah environment.

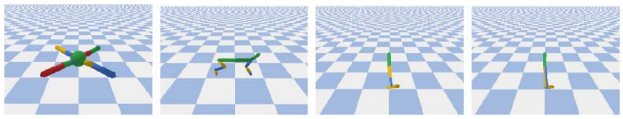


Fig. 11. Pybullet environments.

The potential f value should be largely smaller than the size of the replay buffer. Because the size of the replay buffer in this article is $5e^4$, we test five different f values in Fig. 10. This curve is generated with the same rule used in the sensitive analysis of λ value. The best received mean rewards of each f value are 1881.33, 2014.33, 1884.82, 1775.44, and 1470.74, respectively. Generally, a more frequently updated baseline assists in showing the true changing rate of the current policy. In this article, we select $f = 3000$.

V. Simulation Results

In this article, we use seven different environments freely provided by Pybullet. Four of them are shown in Fig. 11. Benchmarks include A2C, PPO, D4PG, DDPG, SAC, and TD3. Among them, A2C and PPO are popular online methods. Others are popular offline methods. All implementations of the benchmarks are from Maxim Lapan's book [51]. Detailed implementations of the benchmarks can be found in his book. The main parameters of all methods are summarized as follows.

1) For all methods, the actor network and the critic network are composed of three fully connected layers with 256 hidden units, activated with the rectified linear unit (ReLU) and Tanh functions. Specifically, for the actor network, the first layer has state dimension $\times$ hidden size parameters followed by the ReLU function. The second layer has hidden size $\times$ hidden size parameters followed by the Tanh function. The last layer has hidden size $\times$ action dimension parameters followed by the Tanh function. If the actions' standard deviations are needed, another output layer will be added. The critic network has a similar structure. The batch size is 128. $\gamma = 0.95$, $\alpha = 0.005$, and the optimizer used is Adam. Learning rate is $3e^{-4}$. The training steps are $2e^6$, and after each $1e^4$ time steps, we will run ten times of testing.

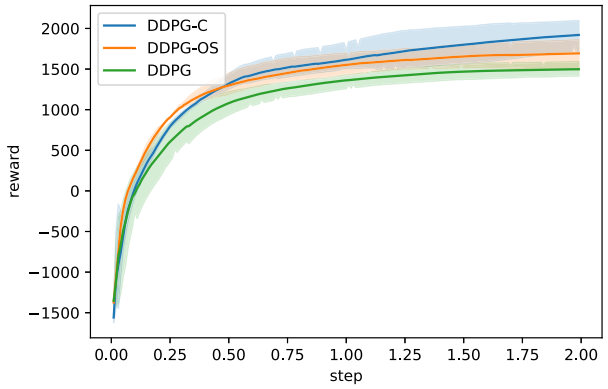


Fig. 12. Performance comparison among our DDPG-C (DDPG with compatible critic network and without overestimation solution), DDPG-OS and DDPG in HalfCheetah environment.

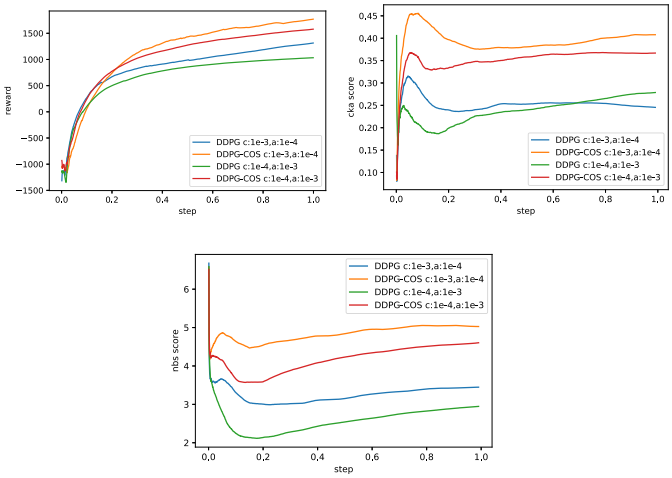


Fig. 13. Influence of the learning rate on the compatibility with DDPG and DDPG-COS. The better performance and similarity scores will be obtained when the critic network is trained with a little bit higher learning rate.

2) For A2C, $\beta = 1e^{-3}$ which is used in the entropy loss as additional regularization term. For PPO, $\varepsilon = 0.2$ is used in the performance function. $\lambda = 0.95$, which is used in the advantage estimator. Moreover, the trajectory size is 2049. For D4PG, the distribution range of $V_{\min} = -10$, $V_{\max} = 10$, and the number of atoms is 51. For TD3, we update the critic network twice at each time step.

Fig. 12 compares the performance of our DDPG-C (DDPG with compatible critic network and without overestimation solution), DDPG with overestimation solution (DDPG-OS), and DDPG in HalfCheetah Environment. The training steps are $2e^6$, and after each $1e^4$ step, we run ten times of testing. The curve plotted shows the mean value of the total rewards of the ten runs of testing. The best mean rewards are 2096.39, 1861.54, and 1591.40 for these three algorithms. We can observe that the compatible critic network boosts the performance of DDPG by 31.73%, and the overestimation solution increases the performance of DDPG by 16.97%.

Learning rate is an important parameter influencing the training process of the neural network. $1e^{-3}$ and $1e^{-4}$ are normal learning rate settings. Fig. 13 demonstrate the influence

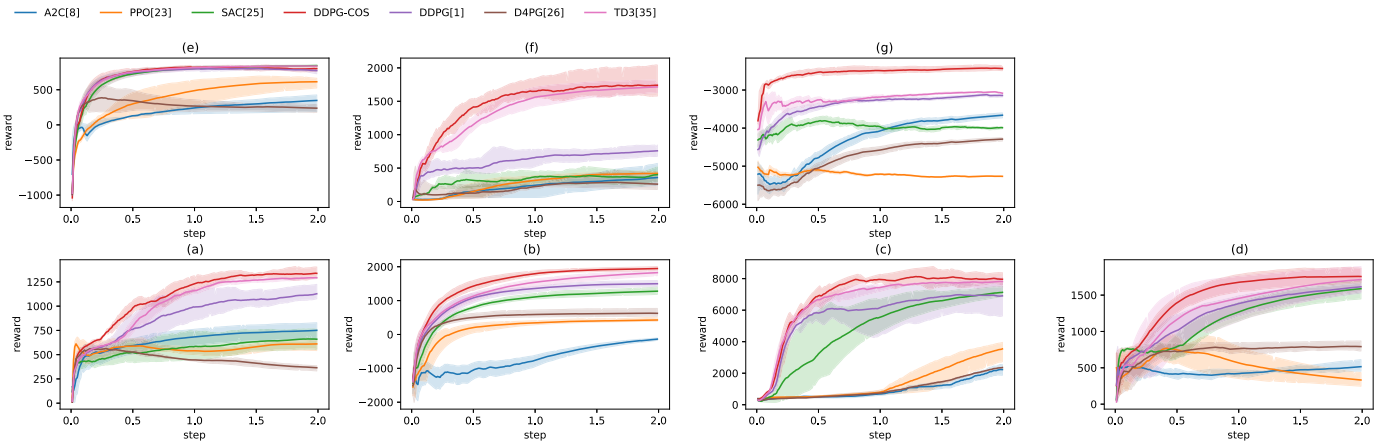


Fig. 14. Performance comparisons among DDPG-COS and other benchmarks in seven environments. (a) Ant. (b) HalfCheetah. (c) InvDoublePen. (d) Walker2d. (e) InvPenSwingup. (f) Hopper. (g) Kuka.

TABLE I
MAXIMUM MEAN REWARDS OF DDPG-COS AND OTHER BENCHMARKS IN SIX ENVIRONMENTS

Environment	A2C[8]	PPO[23]	SAC[25]	DDPG-COS	DDPG[1]	D4PG[26]	TD3[35]
Ant	828.82	681.68	765.86	**1408.22**	1224.19	608.54	1302.70
HalfCheetah	-106.08	505.52	1398.56	**2006.00**	1591.40	768.47	1881.01
InvDoublePen	2546.86	3956.51	7503.41	**8767.34**	7808.91	2363.96	8108.10
InvPenSwingup	425.97	694.72	853.66	**854.49**	824.24	584.58	849.43
Hopper	569.77	449.98	522.05	**2045.31**	845.89	526.52	1819.01
Walker2D	618.37	965.43	1717.06	**1879.93**	1666.12	868.56	1863.19
Kuka	-3602.24	-4866.45	-3682.21	**-2338.85**	-3085.03	-4248.66	-3038.43

of the learning rate on the compatibility with DDPG and DDPG-COS in the HalfCheetah environment. We test two different conditions, setting the learning rate of the critic network and the actor network $1e^{-3}$, $1e^{-4}$, respectively, and vice versa. Fig. 13 is plotted with the testing data. The training steps are $1e^6$, and after each $1e^4$ time steps, we run ten times testing and calculate the similarity indices. In consistent with [35], better performance will be obtained when the critic network is trained with a little bit higher learning rate. Besides, we will get a higher CKA score and NBS score. A faster-converged critic network will provide the actor network with more accurate guidance. Moreover, our proposed DDPG-COS can amend the training process of the critic network with the policy change information. Moreover, we can find that DDPG-COS has higher performance and similarity scores in these testing. Besides, peaks and valleys on the CKA and NBS curves show the challenges of the compatibility issue.

Fig. 14 compares the performance of our algorithm with the other six benchmarks in seven different environments. We run the training process three times. In each training process, the training steps are $2e^6$, and after each $1e^4$, we run ten times of testing. The curve plotted is the mean rewards of 30 runs of testing. As mentioned above, A2C and PPO are online methods that require more training steps. However, $2e^6$ is large enough for the off-line methods, like DDPG or TD3. As is shown in Fig. 14, our algorithm receives the highest mean rewards in all environments at an early stage. Table I shows the maximum mean rewards of all methods in all environments. For example, in the Ant environment, DDPG-COS is 8.10% larger than the second-highest method, TD3, and 15.03% larger than the third-highest method, DDPG. In the HalfCheetah environment, DDPG-COS is 6.65% larger than the second-highest method, TD3, and 26.05% larger than the third-highest method, DDPG. In the Hopper environment, DDPG-COS is 12.44% larger than the highest method, TD3, and 141.79% than the third-largest method, DDPG.

VI. CONCLUSION

Deep RL, especially DDPG, achieves outstanding success in recent years. In this article, we study the critic network's compatibility issue in DDPG, which, to our knowledge, is a less touched topic. Compatibility issues come from the work of Silver *et al.* [14], if a function-approximated critic network is not compatible with the actor network, it cannot provide the true gradient to the actor network. However, the linear compatible function is not suitable for the sizeable state-action space problem. In this article, we first measure the compatibility with the introduced similarity of neural networks with gradient. The actor network and the critic network are represented as kernel matrices considering the feature vectors and gradient vectors of each neural network layer. Then the CKA index and the NBS index provide us with a view of the compatibility between the actor network and the critic network. Besides, we propose a compatible critic network to solve the problem mentioned above. We first rebuild the linear compatible function as an energy function model and introduce this model into the critic

$$\nabla_\omega Q^\omega(s,a) = \delta(s,a)\cdot \text{rate} \tag{28}$$

$$Q^\omega(s,a) = \lim_{\triangle\omega\to 0} \triangle\omega\cdot\delta(s,a)\cdot\text{rate} \tag{29}$$

$$\nabla_{a=\mu_\theta(s)} Q^w(s,a) = \lim_{\triangle\omega\to 0}\triangle\omega\cdot\left(\nabla_{a=\mu_\theta(s)}\delta(s,a)\cdot\text{rate}+\delta(s,a)\cdot\nabla_{a=\mu_\theta(s)}\text{rate}\right) \tag{30}$$

$$= \lim_{\triangle\omega\to 0}\triangle\omega\cdot\left[\nabla_a\delta(s,a)\cdot(c_1+c_2e^{(\mu_\theta(s)-d)^2})+\delta(s,a)\cdot b_1\cdot e^{(\mu_\theta(s)-d)^2}\cdot\nabla_\theta\mu_\theta(s)\right] \tag{31}$$

$$= \lim_{\triangle\omega\to 0}\triangle\omega\cdot\left[\nabla_a Q^\omega(s,a)\cdot(c_1+c_2e^{(\mu_\theta(s)-d)^2})+b_2\cdot e^{(\mu_\theta(s)-d)^2}\cdot\nabla_\theta\mu_\theta(s)\right] \tag{32}$$

$$= \lim_{\triangle\omega\to 0}\triangle\omega\cdot\frac{b_2\cdot e^{(\mu_\theta(s)-d)}}{1-\lim_{\triangle\omega\to 0}\triangle\omega(c_1+c_2e^{(\mu_\theta(s)-d)^2})}\cdot\nabla_\theta\mu_\theta(s) \tag{33}$$

network's optimization process. This article demonstrates the necessity and advantages of the compatible critic network from three aspects: the policy improvement/evaluation steps, the theoretical analysis, and the experiment results. Besides, we prove that our proposed algorithm satisfies the compatible function approximation theorem, which guarantees the convergence and ensures that our compatible critic network can provide the true gradient to the actor network under the function approximation. Moreover, based on our experiments' observations, we propose a light-computation overestimation solution. And we surprisingly find the additional benefits of the compatible critic network for mitigating overestimation. Empirically, we demonstrate our DDPG-COS method surpasses DDPG with consistent higher mean rewards and similarity scores. Besides, we study the sensitivity analysis and the influence of the learning rate on compatibility. Finally, we prove that the performance of our method surpasses state-of-the-art in all seven environments.

In the future, we will extend our method to multiagent RL algorithms. For example, in multi-agent deep deterministic policy gradient (MADDPG), the multiagent setting requires the central critic to instruct all agents. If the central critic network is compatible with a particular leader agent's policy or all agents' policies, it will provide better estimations and boost the overall performance.

Appendix
Proof of the Satisfaction of the Compatible Function Approximation Theorem

In this section, we prove our proposed compatible critic network satisfying the condition (18). In DDPG, assume $\nabla_\omega Q^\omega(s,a) = \delta(s,a)$. With the Taylor series extension, $\delta(s,a) = \lim_{\triangle\omega\to 0}(Q^w(s,a)-Q_0)$. Then in our algorithm, (28)–(33), as shown at the top of the page, where $c_1 = (e^{(a-\mu_\theta(s))^2}/(1/N)\sum_{j=1}^N \varepsilon(s_j,a_j,\theta))$, $c_2 = (1/(1/N)\sum_{j=1}^N \varepsilon(s_j,a_j,\theta))$, $b_1 = (2/(1/N)\sum_{j=1}^N \varepsilon(s_j,a_j,\theta))$, $b_2 = (2\delta(s,a)/(1/N)\sum_{j=1}^N \varepsilon(s_j,a_j,\theta))$ and $d = \mu'_\theta(s)$. Since $1-\lim_{\triangle\omega\to 0}\triangle\omega(c_1+c_2e^{(\mu_\theta(s)-d)^2})\approx 1$, (33) can be rewritten as

$$\nabla_{a=\mu_\theta(s)} Q^w(s,a) = \lim_{\triangle\omega\to 0}\triangle\omega \cdot\delta\cdot\frac{e^{(\mu_\theta(s)-d)}}{\frac{1}{N}\sum_{j=1}^N\varepsilon(s_j,a_j,\theta)}\cdot\nabla_\theta\mu_\theta(s) \tag{34}$$

where δ can be removed by regularizing the gradient in the optimization process of the critic network. $(e^{(\mu_\theta(s)-d)^2}+e^{(\mu_\theta(s)-d)^2})/((1/N)\sum_{j=1}^N\varepsilon(s_j,a_j,\theta))$ is a regularized term. Besides, With a small size experience buffer, $(a-\mu_\theta(s))^2\approx(\mu_\theta(s)-\mu_{\theta'}(s))^2$. If we remove these two regularized terms in (34), we get $\nabla_a Q^\omega(s,a)\mid_{a=\mu_\theta(s)} = \triangle\omega\,\nabla_\theta\,\mu_\theta(s)$, which is the condition in (18).

References

[1] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. ICLR*, Y. Bengio and Y. LeCun, Eds., 2016, pp. 1–14.

[2] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Connectionist Models Summer School*. Hillsdale, NJ, USA: Lawrence Erlbaum, 1993.

[3] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[4] N. Wang, Y. Gao, and X. Zhang, "Data-driven performance-prescribed reinforcement learning control of an unmanned surface vehicle," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 19, 2021, doi: 10.1109/TNNLS.2021.3056444.

[5] X. Yang *et al.*, "Hierarchical reinforcement learning with universal policies for multistep robotic manipulation," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 1, 2021, doi: 10.1109/TNNLS.2021.3059912.

[6] C. Wirth, R. Akrour, G. Neumann, and J. Fürnkranz, "A survey of preference-based reinforcement learning methods," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 4945–4990, 2017.

[7] J. S. Obando-Ceron and P. Samuel Castro, "Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research," 2020, *arXiv:2011.14826*. [Online]. Available: http://arxiv.org/abs/2011.14826

[8] P. Hummel and R. P. McAfee, "Machine learning in an auction environment," *The J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 6915–6951, 2016.

[9] D. Martinez, G. Alenya, T. Ribeiro, K. Inoue, and C. Torras, "Relational reinforcement learning for planning with exogenous effects," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2689–2732, 2017.

[10] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[11] T. D. Bruin, J. Kober, K. Tuyls, and R. Babuška, "Experience selection in deep reinforcement learning for control," *J. Mach. Learn. Res.*, vol. 19, no. 1, pp. 347–402, 2018.

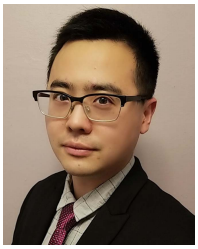
[12] N. Heess, D. Silver, and Y. W. Teh, "Actor-critic reinforcement learning with energy-based policies," in *Proc. Eur. Workshop Reinforcement Learn.*, 2013, pp. 45–58.

[13] K. Ciosek and S. Whiteson, "Expected policy gradients for reinforcement learning," *CoRR*, vol. abs/1801.03326, pp. 1–51, Jan. 2018.

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. ICML*, in JMLR Workshop and Conference Proceedings, vol. 32. JMLR.org, 2014, pp. 387–395. [Online]. Available: http://proceedings.mlr.press/v32/silver14.html and https://www.bibsonomy.org/bibtex/2e2fb52847293919f2e6c88fc8c9eee9b/dblp

[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

[16] A. B. Labao, M. A. M. Martija, and P. C. Naval, "A3C-GS: Adaptive moment gradient sharing with locks for asynchronous actor–critic agents," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 1162–1176, Mar. 2021.
[17] R. J. Williams and J. Peng, "Function optimization using connectionist reinforcement learning algorithms," *Connection Sci.*, vol. 3, no. 3, pp. 241–268, 1991.
[18] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, "Understanding the impact of entropy on policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 151–160.
[19] C. Cortes, M. Mohri, and A. Rostamizadeh, "Algorithms for learning kernels based on centered alignment," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 795–828, Jan. 2012.
[20] B. Muzellec and M. Cuturi, "Generalizing point embeddings using the wasserstein space of elliptical distributions," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10258–10269.
[21] S. M. Kakade, "A natural policy gradient," in *Proc. Adv. Neural Inf. Process. Syst.*, 2002, pp. 1531–1538.
[22] 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.
[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, abs/1707.06347, pp. 1–12, Jul. 2017.
[24] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5279–5288, 2017.
[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, *arXiv:1801.01290*. [Online]. Available: http://arxiv.org/abs/1801.01290
[26] G. Barth-Maron *et al.*, "Distributed distributional deterministic policy gradients," in *Proc. ICLR*, 2018, pp. 1–16.
[27] R. S. Sutton, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1057–1063.
[28] R. S. Sutton *et al.*, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML)*, 2009, pp. 993–1000.
[29] D. Balduzzi and M. Ghifary, "Compatible value gradients for reinforcement learning of continuous deep policies," *CoRR*, vol. abs/1509.03005, pp. 1–16, Apr. 2015.
[30] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proc. 3rd IEEE-RAS Int. Conf. Hum. Robot.*, Oct. 2003, pp. 1–20.
[31] A. Achille *et al.*, "Task2 Vec: Task embedding for meta-learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6430–6439.
[32] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3519–3529.
[33] S. Tang, W. J. Maddox, C. Dickens, T. Diethe, and A. Damianou, "Similarity of neural networks with gradients," 2020, *arXiv:2003.11498*. [Online]. Available: http://arxiv.org/abs/2003.11498
[34] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[35] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, *arXiv:1802.09477*. [Online]. Available: http://arxiv.org/abs/1802.09477
[36] D. Wu, X. Dong, J. Shen, and S. C. Hoi, "Reducing estimation bias via triplet-average deep deterministic policy gradient," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4933–4945, Nov. 2020.
[37] L. Meng, R. Gorbet, and D. Kulić, "The effect of multi-step methods on overestimation in deep reinforcement learning," 2020, *arXiv:2006.12692*. [Online]. Available: http://arxiv.org/abs/2006.12692
[38] L. Pan, Q. Cai, and L. Huang, "Softmax deep double deterministic policy gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–17.
[39] Z. Zhang, Z. Pan, and M. J. Kochenderfer, "Weighted double Q-learning," in *Proc. IJCAI*, 2017, pp. 3455–3461.
[40] O. Anschel, N. Baram, and N. Shimkin, "Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 176–185.
[41] D. Lee, B. Defourny, and W. B. Powell, "Bias-corrected Q-learning to control max-operator bias in Q-learning," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn. (ADPRL)*, Apr. 2013, pp. 93–99.
[42] S. Frank He, Y. Liu, G. Alexander Schwing, and J. Peng, "Learning to play in a day: Faster deep reinforcement learning by optimality tightening," *CoRR*, vol. abs/1611.01606, pp. 1–13, Nov. 2016.
[43] O. Nachum, M. Norouzi, G. Tucker, and D. Schuurmans, "Smoothed action value functions for learning Gaussian policies," *CoRR*, vol. abs/1803.02348, pp. 1–12, Mar. 2018.
[44] W. Shi, S. Song, C. Wu, and C. L. P. Chen, "Multi pseudo Q-learning-based deterministic policy gradient for tracking control of autonomous underwater vehicles," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 12, pp. 3534–3546, Dec. 2019.
[45] K. D. Asis, J. F. Hernandez-Garcia, G. Z. Holland, and A. S. Sutton, "Multi-step reinforcement learning: A unifying algorithm," in *Proc. 32nd AAAI Conf. on Artif. Intell.*, 2018, pp. 2902–2909.
[46] S. Al-Dabooni and D. C. Wunsch, "An improved N-step value gradient learning adaptive dynamic programming algorithm for online learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1155–1169, Apr. 2020.
[47] S. Al-Dabooni and D. Wunsch, "The boundedness conditions for model-free HDP (λ)," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 7, pp. 1928–1942, Jul. 2018.
[48] T. Degris, M. White, and S. Richard Sutton, "Off-policy actor-critic," *CoRR*, vol. abs/1205.4839, pp. 1–18, Mar. 2012.
[49] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *CoRR*, vol. abs/1702.08165, pp. 1–16, Feb. 2017.
[50] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning," in *Proc. AAAI*, 1993, pp. 99–107.
[51] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, With Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Birmingham, U.K.: Packt, 2018.



Di Wang (Student Member, IEEE) received the B.S. degree in electrical engineering from Fuzhou University, Fuzhou, China, in 2014, and the M.S. degree in electrical engineering from Tianjin University, Tianjin, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Industrial Engineering, University of Illinois at Chicago, Chicago, IL, USA.

His current research interests include multiagent systems, distributed control, and energy schedule in the smart city.

Mengqi Hu (Member, IEEE) received the Ph.D. degree in industrial engineering from Arizona State University, Tempe, AZ, USA, in 2012.

He is currently an Assistant Professor with the Department of Mechanical and Industrial Engineering, University of Illinois at Chicago, Chicago, IL, USA. He has published more than 30 articles in journal such as the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Information Sciences*, and *Applied Energy*. His current research interests include multiagent decision making and reinforcement learning with applications in autonomous vehicles and smart grid.

Dr. Hu serves as an Associate Editor for *Swarm and Evolutionary Computation* Journal.