Knowledge-Based Systems 194 (2020) 105511

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Obtaining accurate estimated action values in categorical distributional reinforcement learning☆

Yingnan Zhao, Peng Liu, Chenjia Bai, Wei Zhao *, Xianglong Tang

Harbin Institute of Technology, Harbin 150001, China

ARTICLE INFO

Article history:
Received 3 July 2019
Received in revised form 8 January 2020
Accepted 10 January 2020
Available online 18 January 2020

MSC:
00-01
99-00

Keywords:
Distributional reinforcement learning
Estimated action value
Bootstrapping
Interval estimation

ABSTRACT

Categorical Distributional Reinforcement Learning (CDRL) uses a categorical distribution with evenly spaced outcomes to model the entire distribution of returns and produces state-of-the-art empirical performance. However, using inappropriate bounds with CDRL may generate inaccurate estimated action values, which affect the policy update step and the final performance. In CDRL, the bounds of the distribution indicate the range of the action values that the agent can obtain in one task, without considering the policy's performance and state–action pairs. The action values that the agent obtains are often far from the bounds, and this reduces the accuracy of the estimated action values. This paper describes a method of obtaining more accurate estimated action values for CDRL using adaptive bounds. This approach enables the bounds of the distribution to be adjusted automatically based on the policy and state–action pairs. To achieve this, we save the weights of the critic network over a fixed number of time steps, and then apply a *bootstrapping* method. In this way, we can obtain confidence intervals for the upper and lower bound, and then use the upper and lower bound of these intervals as the new bounds of the distribution. The new bounds are more appropriate for the agent and provide a more accurate estimated action value. To further correct the estimated action values, a distributional target policy is proposed as a smoothing method. Experiments show that our method outperforms many state-of-the-art methods on the OpenAI gym tasks.

1. Introduction

The goal of reinforcement learning (RL) is to learn an optimal policy for sequential decision-making problems, by interacting with the environment [1]. Recent research has shown that the integration of RL and deep neural networks can solve problems in a wide range of fields [2], including games [3–5], robotics [6–9], natural language [10,11] and computer vision [12,13].

One of the major classes of RL is value-based methods, which learn the optimal value function using the Bellman operator. Almost every RL algorithm is based on this approach. The Deep Deterministic Policy Gradient (DDPG) [14] extends two other important algorithms, the deep Q-network and deterministic policy gradient, which estimate the policy gradient more efficiently and stabilize the learning. The Asynchronous Advantage Actor Critic (A3C) [15] learns a value function and a policy. The value function reduces variance and accelerates learning; A3C can also run several agents in parallel to reduce the training time. Both DDPG and A3C exhibit good performance on complex control tasks with high-dimensional action spaces.

Categorical Distributional RL (CDRL) [16] views RL problems from a whole new perspective. In contrast to the value-based algorithms, which learn the expectation of the action value, CDRL models the full distribution of the action value. CDRL preserves the multimodality of the value distributions, leading to more stable learning and better policies than other methods. CDRL uses a categorical distribution with evenly spaced outcomes, and the target value distribution can be computed through a distributional Bellman operator. However, the target and the predicted distribution have disjoint supports. To account for this, CDRL projects the target distribution onto the supports of the prediction, and then applies a Kullback–Leibler minimization step.

CDRL assigns probabilities to an a priori fixed, discrete set of possible returns, which is determined by the number of atoms l and the bounds (Q_{min}, Q_{max}). The number of possible returns that the agent can obtain is l and the distance between two atoms is $\Delta = \frac{Q_{max}-Q_{min}}{l-1}$. According to distributional RL theory [16], the expectation of this return distribution (estimated action value) should be approximately equal to the true action value, otherwise

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2020.105511.

* Corresponding author.
E-mail addresses: ynzhao_atari@hit.edu.cn (Y. Zhao), pengliu@hit.edu.cn (P. Liu), bai_chenjia@stu.hit.edu.cn (C. Bai), zhaowei@hit.edu.cn (W. Zhao), tangxl@hit.edu.cn (X. Tang).

https://doi.org/10.1016/j.knosys.2020.105511
0950-7051/

the performance will be affected. The estimated action value impacts the update of the distribution, as well as the estimated policy gradient. If it is not accurate, the performance of the policy will suffer.

The bounds of this distribution impact the estimated action value and play an important role in CDRL. (Q_{min}, Q_{max}) and l define the possible action values that the agent can get. In CDRL, (Q_{min}, Q_{max}) is set as the minimal and maximal expected cumulative reward the agent can receive in each task, and remains fixed throughout the training. The bounds are set to be wide enough to cover all the possible action values. However, most of the time, the performance of the policy is not as good as the optimal policy. Therefore, the true maximum action value that the agent can achieve is far less than Q_{max}. As a result, the atoms between the true maximum action value and the upper bound Q_{max} will have some estimation error. These bounds may also be too wide for those state–action pairs that cannot achieve large action values, and so the estimated action value will be inaccurate. To make the estimated action value more accurate, the bounds should be adjusted automatically for different learning stages and state–action pairs.

In this study, to make the estimated action value more accurate and improve the CDRL performance, we developed adaptive bounds that can be adjusted depending on the performance of the policy and the different state–action pairs. To achieve this, we train a critic network that can produce the same action value as common value-based RL methods. Then, we save the weights of this critic network over a fixed number of time steps, allowing the action values to be used as the samples for different state–action pairs in different learning stages. Finally, we estimate a confidence interval for Q_{max} and Q_{min} based on these samples. When calculating the confidence interval, the sample dependencies, changing policies, and non-stationary and limited sample size create certain challenges. Thus, we use a moving block bootstrap [17,18]. We construct several blocks to eliminate the dependencies between samples and then we sample the blocks using *bootstrapping* [19]. These bootstrapped samples can be used to approximate a 1-α confidence interval around the bounds of the categorical distribution. Therefore, the bounds can be adjusted automatically for different state–action pairs and learning stages, resulting in a more accurate approximation of the action value and improved performance. To reduce the variance in the update and make the estimated action value more accurate, we also developed a distributional target policy smoothing (DTPS) strategy. The key idea behind DTPS comes from Expected SARSA [20]. DTPS computes the target policy based on the expected distribution, by adding a small random noise to the target action and averaging their distributions. To evaluate our method in continuous-control tasks, we combine our improved CDRL with DDPG. Experiments demonstrate that this configuration achieves state-of-the-art performance.

The main contributions of this paper are as follows: (1) We show that the problem of inaccurate estimated action values persists in CDRL. (2) Adaptive bounds are proposed to make the estimate action value more accurate. (3) To further correct the estimated action value, we propose a distributional target policy smoothing strategy. (4) Evaluations on several continuous-control tasks demonstrate that our method can lead to better performance.

The remainder of this paper is structured as follows. Section 2 provides a brief introduction to RL and methods related to the topic of this paper. Our approach is presented in Section 3, before the results of experiments are presented and discussed in Section 4. Finally, Section 5 summarizes the conclusions to this study.

2. Background

2.1. Reinforcement learning

RL solves sequential decision making problems with the aim of learning reward-maximizing behavior [1,21]. A Markov decision process (MDP) can be used as a formalization of RL problems. In MDP, the agent is given a state $s \in S$ at each discrete time step and selects an action $a \in A$ following the policy $\pi : S \to A$. Once the action is executed, the agent receives a reward $r(s, a)$ and the environment moves to a new state s'. The total return is defined as follows:

$$G_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \tag{1}$$

where $\gamma \in (0, 1]$ is the discount factor, which determines the priority of short-term rewards. At each time step, the agent can use experience of the form $(s, a, s', r(s, a))$ to improve its policy and maximize the expectation of G_t.

Value-based methods are effective for solving RL problems. The key component of value-based methods is the action value function. The action value function $Q^{\pi}(s, a) = \mathrm{E}[G_t | s_t = s, a_t = a, \pi]$ describes the expected return from selecting action a in state s and then acting according to π. We say that the action value is the expectation or the mean of the value distribution. Q-learning [22–24] is a central algorithm in the field of RL. The update rule for this approach is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{2}$$

By iterating rule (2), Q-learning converges to the optimal action value function [21], from which an optimal policy can be derived. For a large state space, Mnih et al. introduced the Deep Q-Network (DQN) [4], which uses deep neural networks to approximate the action value function. To further stabilize the training process, DQN uses a target network and experience replay.

In contrast to value-based methods such as Q-learning, policy-based methods directly optimize the parameterized policy $\pi(a|s; \theta)$. REINFORCE [25,26] is a policy-based method that updates θ in the direction of $\nabla_{\theta} log \pi(a_t|s_t, \theta)(G_t - b_t(s_t))$, where $b_t(s_t)$ is used as a baseline to reduce the variance of the gradient estimate.

2.2. Categorical distributional reinforcement learning

CDRL models the entire distribution of returns, rather than just the expected returns. Empirically learning the distribution over returns improves data efficiency, final performance, and stability [16,27,28].

Obtaining the action value under policy π is a basic problem in RL. One common way to solve this is to use dynamic programming through the Bellman operator [29]:

$$\mathscr{T}^{\pi} Q(s, a) = \mathbb{E}[r(s, a)] + \gamma \mathbb{E}_{P,\pi}[Q^{\pi}(s', a')] \tag{3}$$

where P is the transition kernel $P(.|s, a)$. Similarly, a distributional Bellman operator is defined to compute the value distribution through dynamic programming:

$$\mathscr{T}^{\pi} Z(s, a) :\stackrel{D}{=} r(s, a) + \gamma P^{\pi} Z(s', a') \tag{4}$$

where P^{π} is the transition operator and $Y :\stackrel{D}{=} U$ denotes that the random variable Y is distributed according to the same law as U. The distribution over returns is denoted by Z and $Q = \mathbb{E}[Z]$. A common parametric distribution is the categorical distribution, which is supported on fixed locations $z_1 \leq \cdots \leq z_N$. The parameters of this distribution are the probabilities q_i corresponding

to each location z_i. Similar to value-based methods, the goal is to minimize the loss between the target distribution $\mathscr{T}Z_\theta$ and the current estimated distribution Z_θ. The problem is that $\mathscr{T}Z_\theta$ and Z_θ always have disjoint supports, which makes it difficult to calculate the loss. Thus, CDRL applies a projection step to map the target distribution onto the same supports as Z_θ. Given a sample transition (s, a, r, s'), the Bellman update for one atom z_i can be computed as $\mathscr{T}z_i := r+\gamma z_i$. The associated probability $p_i(s', \pi(s'))$ is then distributed to the immediate neighbors of $\mathscr{T}z_i$. The ith component of the projected update $\Phi\mathscr{T}Z_\theta(s, a)$ is:

$$(\Phi\mathscr{T}Z_\theta(s,a))_i = \sum_{j=1}^{N-1}[1 - \frac{|\lfloor\mathscr{T}z_j\rceil_{Q_{min}}^{Q_{max}} - z_i|}{\Delta z}]_0^1 p_j(s', \pi(s')) \tag{5}$$

We then calculate the sample loss $\mathscr{L}_{s,a}(\theta)$ as follows:

$$\mathscr{L}_{s,a}(\theta) = D_{KL}(\Phi\mathscr{T}Z_\theta(s,a) \parallel Z_\theta(s,a)) \tag{6}$$

Through minimizing the above loss function, CDRL converges to the optimal value distribution and obtains the optimal policy [30, 31].

The bounds (Q_{min}, Q_{max}) of the categorical distribution in CDRL define the range of all the possible returns the agent can attain throughout the training and affect the estimated action value (the estimated action value is the expectation of the categorical distribution). The bounds also affect the projection step, as shown by (5). If the estimated action value is as close as possible to the real action value, CDRL can achieve good performance. Q_{max} and Q_{min} reflect the maximum and minimal cumulative rewards that the agent can obtain on each task. In CDRL, Q_{min} can be set as the cumulative reward that the agent achieves by following the worst policy (random policy), whereas Q_{max} is the cumulative reward that the agent receives following the optimal policy. Thus, Q_{max} is always far greater then Q_{min}. Most of the time, Q_{max} is far above the true maximum action value that the policy can obtain, and the true action value of some state–action pairs can will never get close to Q_{max}. This leads to an inaccurate estimated action value and harms the RL performance.

In this paper, to make the estimated action value more accurate, we describe the construction of adaptive bounds for CDRL. These can be adjusted for different courses of training and state–action pairs by using the bootstrapping method to approximate the confidence interval of the bounds. We use the upper and lower bounds of these confidence intervals as the new bounds of the distribution. Finally, we introduce the DTPS method to further reduce the variance of the update and correct the estimated action value.

2.3. Deep deterministic policy gradients

DDPG is an actor–critic, model-free method for continuous action space, that extends DQN [4] and the deterministic policy gradient (DPG) [32]. DDPG deploys an experience replay technique and a target network to stabilize the training. The core element of DDPG is the DPG theorem and the gradient of the actor, which can be written as follows:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_\rho[\nabla_\theta\pi_\theta(x)\nabla_a Q_{\pi_\theta}(x,a)|_{a=\pi_\theta(x)}] \tag{7}$$

where ρ is the state-visitation distribution. DDPG maintains an estimate of the action value function $Q_\omega(s, a)$ by minimizing the temporal difference error between the action value function before and after applying the Bellman update. Therefore, we can write the resulting loss as:

$$L(\omega) = \mathbb{E}_\rho[(Q_\omega(s,a) - \mathscr{T}^\pi Q_{\omega'}(s,a))^2] \tag{8}$$

Barth-Maron et al. [33] proposed the distributional deep deterministic policy gradient (D3PG), which combines CDRL with DDPG and has achieved state-of-the-art performance in several complex tasks. D3PG considers the inclusion of a distributional critic [16] and adapts CDRL to continuous control setting. To use the distributional critic within the context of the DDPG architecture introduced above, we need to rewrite the DPG theorem by taking the expectation with respect to the action-value distribution, i.e.,

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \mathbb{E}_\rho[\nabla_\theta\pi_\theta(x)\nabla_a Q_\omega(x,a)|_{a=\pi_\theta(x)}] \\ &= \mathbb{E}_\rho[\nabla_\theta\pi_\theta(x)\mathbb{E}[\nabla_a Z_\omega(x,a)]|_{a=\pi_\theta(x)}]\end{aligned} \tag{9}$$

D3PG updates the critic in the same way as CDRL, including a projection step and a KL minimization step. The inclusion of the distributional critic update means that D3PG is capable of state-of-the-art performance on a number of difficult continuous problems.

2.4. Related work

CDRL is a new perspective of RL: when combined with DQN, it outperforms other algorithms that only model the expected values in of many Atari 2600 games [34,35]. Besides Categorical distribution, the paper [33] proposed a method to model the return distribution with Mixture of Gaussians distribution, which can reduce the requirement of domain-knowledge and remove the projection step, but performs worse than categorical distribution. MoG-DQN [36] modeled the return distribution with Mixture of Gaussian distribution and proposed a new distance metric for MoG distribution: Jensen–Tsallis Distance (JTD). The advantage of JTD over other metrics such as KL-Divergence and the Wasserstein Distance is that JTD computes the distance between two MoG distributions in a closed form, which makes JTD loss applicable with sample-based methods.

More recently, additional distributional algorithms have been proposed. Quantile regression (QR-DQN) [37] does not require the projection step and can perform distributional RL over the Wasserstein metric [38]. The Implicit Quantile Networks (IQN) technique [39] uses quantile regression to approximate the full quantile function for the return, and produces risk-sensitive policies. Bellemare et al. [40] provided theoretical and empirical results that explain why distributional RL produces the observed improvements. Dopamine [41] is a code repository containing implementations of many distributional RL algorithms. Tang et al. [42] propose a framework based on distributional RL and unifies several methods in exploration. Dabney et al. [31] present a unifying framework for designing and analyzing distributional RL and provide improved analyses of existing distributional RL. Tamar et al. [43] show that the distributional Bellman equation is equivalent to a generative adversarial network (GAN) model. The authors use this insight to propose a GAN-based approach and can be used in the multivariate rewards setting.

Our work is also related to solving continuous control tasks. Proximal Policy Optimization (PPO) takes the biggest possible improvement step on a policy without stepping so far. PPO methods are simpler to implement and performs better than other policy gradient methods. Guide Actor–Critic (GAC) [44] utilizes Hessians of the critic for actor learning and learns a guide actor that locally maximizes the critic. In Dual Actor–Critic (Dual-AC) [45], the actor and dual critic are updated cooperatively to optimize the same objective function. Trust Path Consistency Learning (Trust-PCL) [46] is an off-policy algorithm employing a relative-entropy penalty to impose a trust region on a maximum reward objective and can perform well on a set of standard control tasks. Soft Actor–Critic (SAC) [47] is a new RL framework, in SAC the actor aims to maximize expected reward while also maximizing entropy to succeed at the task while acting as randomly as possible. Twin Delayed Deep Deterministic policy gradient (TD3) [48] increase the stability and performance with consideration of function approximation error.

3. Proposed method

In this section, we first show that inappropriate CDRL bounds result in an estimation error, and that this estimation error affects the overall performance. In Section 3.2, we describe an adaptive bounds method using bootstrapping that addresses this problem. A distributional target policy smoothing strategy is presented in Section 3.3. Section 3.4 summarizes the whole algorithm.

3.1. Motivation

In CDRL, the categorical distribution has an important hyperparameter: the bounds of the support (Q_{min}, Q_{max}). These bounds determine the range of the possible action value that the agent can attain throughout the training process and the estimated action value. The action value depends on the policy's performance and the state–action pairs. The bounds (Q_{min}, Q_{max}) in CDRL only consider the minimum and maximum action value the agent can obtain, and remain fixed throughout the training stages. Thus, the bounds are not accurate enough for the estimation of the action value.

Given the bounds (Q_{min}, Q_{max}) and the number of atoms l, we assume the support set of this distribution is $\{z_i = Q_{min} + i\Delta z : 0 \le i \le l\}$, $\Delta z := \frac{Q_{max}-Q_{min}}{l-1}$. The atom probabilities are given by a parametric model $\theta : S \times A \to \mathbb{R}^l$. We assume the probability of each atom z_i for state–action pair (s, a) is $p_i(s, a; \theta)$. The initial distribution is typically uniform, as shown in Fig. 1(a), so we obtain: $p_i(s, a; \theta) \approx 1/l, i = 1, \ldots, l$. In this way, the action value that the distribution approximates during the early stage is as follows:

$$\begin{aligned}\hat{Q}(s,a) &= \sum_{i=1}^{l} z_i * p_i(s,a;\theta) \\ &\approx \sum_{i=1}^{l} z_i * \frac{1}{l} \\ &= \frac{1}{l} * \sum_{i=1}^{l} z_i \\ &= \frac{1}{l} * \frac{l*(Q_{min}+Q_{max})}{2} \\ &= \frac{(Q_{min}+Q_{max})}{2}\end{aligned} \tag{10}$$

This equation shows that, during the early stage of training, the estimated action value is approximately equal to $\frac{(Q_{min}+Q_{max})}{2}$. This is greater than the real action value that the agent can obtain because the performance of the policy is poor during the early stages of training and the bounds will cause an overestimated bias.

As the training progresses, the estimated action value $\hat{Q}(s, a)$ will decrease and become closer to the real action value. However, because of the inaccurate bounds and the projection step, CDRL will soon underestimate the action value. The parametric distribution is updated by applying the distributional Bellman operator to each atom z_i: $z_i' \leftarrow r + \gamma z_i$, where z_i' is the target atom, $\gamma \in (0, 1]$ is the discount factor, and r is the reward. The relationship between z_i and z_i' is important. We assume that z_i is greater than z_i':

$$\begin{aligned} & r + \gamma z_i \le z_i \\ \Longrightarrow \quad & r \le (1-\gamma) z_i \\ \Longrightarrow \quad & r \le (1-\gamma)(Q_{min} + i\Delta z)\end{aligned} \tag{11}$$

In other words, if $r \le (1-\gamma)(Q_{min} + i\Delta z)$, z_i is greater than z_i'; otherwise z_i, is less than z_i'. Because the distance Δz is large (When Q_{max} is far larger than Q_{min}), inequality (11) will hold easily as i becomes larger. As shown in Fig. 1(b), the target atoms z_3' and z_4' are less than the original atoms z_3 and z_4. After the Bellman update, CDRL applies a projection step. The probability of the target atom will be allocated to the two adjacent atoms, as Fig. 1(b) shows. For example, the probability of the target atom z_4' is p_4'. We define $b = (z_4' - Q_{min})/\Delta z$ and $p_4 = p_4'(b - \lfloor b \rfloor)$. The projection step and the inaccurate bounds mean that the probability of obtaining the higher action value decreases and the whole distribution shrinks, as shown by Fig. 1(c). As a result, the estimated action value will not change any more and the learning stops.

Does the problem of inaccurate estimated action values occur in practice for CDRL? Fig. 2 plots the expected action value of CDRL and the true action value over the learning process on the OpenAI gym environments Ant-v2. The true action value was obtained by averaging the discounted return over 10 000 time-steps following the current policy. Then, we saved the visited state–action pairs during evaluation to obtain the expected value (the expectation of the return distribution). From Fig. 2(a), for the Categorical distribution with fixed bounds, there is an initial overestimated bias between the expected action value and the true action value; then the expected action value changes slowly and the policy gradually stops learning. On the contrary, the expected action value obtained by adaptive bounds is closer to the true action value than that of fixed bounds. Moreover, using adaptive bounds achieves better performance because the achieved true value is larger.

Finally, we demonstrate that the inaccurate estimated action value caused by CDRL harms the overall performance. The error will propagate because of the distributional Bellman operator, which means it learns an action value distribution from another distribution that already has errors. Therefore, the error propagates throughout the whole course of learning. Additionally, in an actor–critic setting, the policy will become poor through the interplay between the actor and the critic. For example, the distributional policy gradient algorithm in D3PG is $\nabla_\theta J(\theta) \approx \mathbb{E}_\rho[\nabla_\theta \pi_\theta(x) \mathbb{E}[\nabla_a Z_w(x, a)]|_{a=\pi_\theta(x)}]$. If the value estimate Z_w itself is inaccurate, then the actor network updates in the wrong direction and becomes poor.

3.2. Adaptive bounds

This section introduces a novel method that automatically adjusts the bounds in CDRL and makes the estimated action value more accurate. The estimated action value is the expectation of the distribution, which is determined by the number of atoms l and the bounds of the distribution. However, l cannot change the expectation as much as the bounds. Thus, we focus on finding the appropriate bounds.

The new bounds are $(Q_{min}(s_t, a_t), Q_{max}(s_t, a_t))$, which can be adjusted based on the policy's performance (time step) and state–action pairs instead of remaining fixed. We cannot attain the new bounds directly, and the only thing we can do is collecting the action values that the agent obtained during several previous time steps and computing a confidence interval for $Q_{min}(s_t, a_t)$ and $Q_{max}(s_t, a_t)$. However, the state–action pairs cannot be the same in continuous-control tasks and we cannot have several action values for one state–action pair. We assume the action value function is parameterized by a critic network Q_θ. During the course of the training, we save a sequence of changing weights $\{\theta_i, \theta_{i+1}, \ldots, \theta_j\}$ from time step i to j. Therefore, we can obtain samples for each state–action pair at different time steps. To estimate the confidence interval, we are still faced with some challenges. The first is the limited sample size, because we cannot

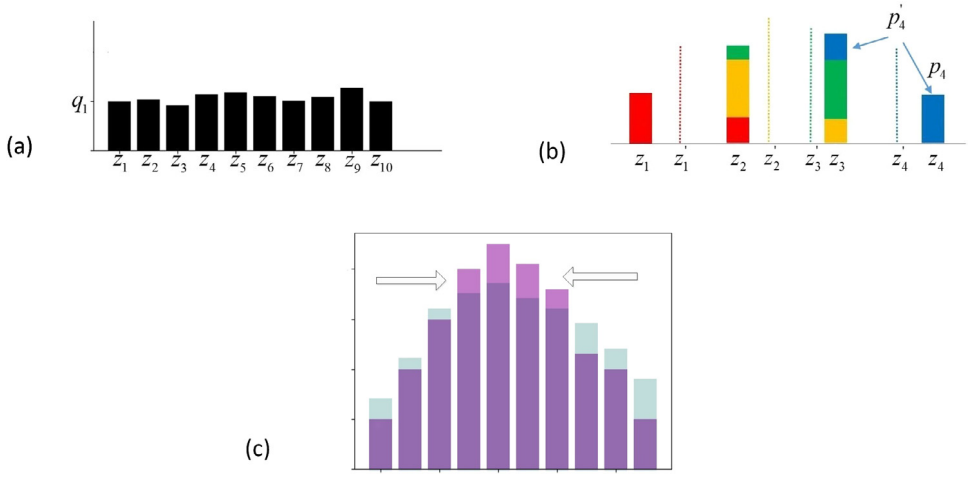
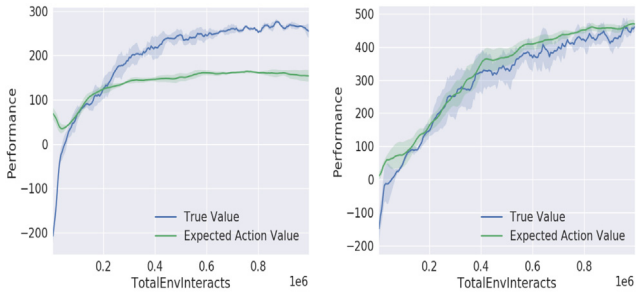


Fig. 1. *Procedure of CDRL*: (a) At the beginning of the training stage, the categorical distribution is uniform. We use $z_1, \ldots, z_{10}$ to denote the possible returns and q_1 is the corresponding probability. (b) Projection step of CDRL. $z_1', \ldots, z_4'$ are the possible returns after applying the distributional Bellman operator and p_4' is the probability of the target position z_4'. The colored dashed lines are the target probabilities. The rectangles with different colors are the results after projection. (c) After projection, the distribution (light color) shrinks to the target distribution (dark color). As a result, the estimated action value will stop updating and the learning process slows down.



(a) Fixed Bounds (b) Adaptive Bounds

Fig. 2. Expected action value and true action value of fixed bounds and our proposed adaptive bounds on Ant-v2.

save too many weights for reasons of computation time and resource. The second issue is the sample dependencies [18], because the weights are strongly interrelated.

The moving block bootstrap (MBB) [17,49,50] technique is known to be valid for this situation. Every bootstrapping method, including MBB, is derived from Efron's original *bootstrapping* procedure [19]. The *bootstrapping* produces a bootstrapped sample from the initial samples, whereas MBB first divides the original sequential data into blocks from which blocks of samples are drawn with replacement. In this way, MBB breaks the dependencies between the sequential data and takes full advantage of the limited number of samples. Finally, these samples are used to estimate the confidence interval of the bounds.

The whole process of estimating the confidence interval is as follows. Given a sequence of changing weights $\{\theta_i, \ldots, \theta_j\}$ from time step i to j, we compute the sample action values $Q_s = \{Q_i(s,a), \ldots, Q_j(s,a)\}$ for one state–action pair (s,a) through the critic network. Then, we construct a set of overlapping blocks $\{B_1, \ldots, B_N\}$, where $N = j-i-l$ is the number of blocks and $B_i = (Q_i(s,a), \ldots, Q_{i+l-1}(s,a))$. $B_1^*, \ldots, B_k^*$ represents a random sample with repetitions from the set $\{B_1, \ldots, B_N\}$. Finally, we merge these blocks to obtain the observations $Q_s^* = \{Q_1^*(s,a), \ldots, Q_{k*l}^*(s,a)\}$. Given these observations, an estimate T_n^* of the statistic can be computed. This process is repeated K times, resulting in K samples of the statistic, $T_{n,1}^*, \ldots, T_{n,K}^*$. With these bootstrapped samples, we construct a $percentile-t(studentized) interval$ as follows:

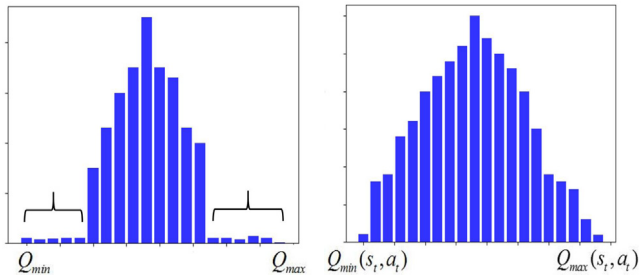
$$P(T \in (2T_n - T_{1-\alpha/2}^*, 2T_n - T_{\alpha/2}^*)) \geq 1-\alpha \tag{12}$$

where T_β^* is the β-quantile of the ordered population and T_n is the statistic of the original sample Q_s. In *bootstrapping*, the β-quantile is usually computed as follows:

$$T_\beta^* = (1-r)T_j^* + rT_{j+1}^* \tag{13}$$

where $j = \lfloor n\beta \rfloor + m$, $r = n\beta - j + m$ and m depends on the quantile type. For non-normal distributions, $m = \frac{\beta+1}{3}$. In this way, we can approximate a 1-α confidence interval $(2T_n - T_{1-\alpha/2}^*, 2T_n - T_{\alpha/2}^*)$, which means the probability of seeing the statistic outside of the interval is low. When $T_n^* = max(Q_s^*)$ and $T_n = max(Q_s)$, we can determine the 1-α confidence interval for the upper bound of the distribution. If $T_n^* = min(Q_s^*)$ and $T_n = min(Q_s)$, we can determine the confidence interval for the lower bound of the distribution. Here the confidence interval for Q_{min} is denoted as $(Q_{min}^{lower}, Q_{min}^{upper})$ and the confidence interval for Q_{max} is denoted as $(Q_{max}^{lower}, Q_{max}^{upper})$. Then we use the upper bound of the confidence interval of Q_{max} as the upper bound of the new distribution, and use the lower bound of the confidence interval of Q_{min} as the lower bound of the new distribution. This means the new bounds of the distribution is $(Q_{min}^{lower}, Q_{max}^{upper})$. In this way, the probability of seeing the true action values inside of the new interval is maximized, which benefits the approximation of the return distribution.

Before updating the policy, we compute the bounds for every state–action pair and replace the oldest weights in the buffer $\{\theta_i, \ldots, \theta_j\}$ by the latest weights every a fixed number of time steps. In this way, the bounds of the distribution adapt to the policy and state–action pairs. At the same time, the bounds will be not too wide (Fig. 3(b)). In Fig. 3(a), the agent cannot obtain action values as large as (or even close to) Q_{max} using CDRL (fixed bounds). Therefore, the atoms between the true maximum action value and Q_{max} (we use braces to denote them in Fig. 3(a)) produce an estimation error. The new bounds reduce the distance between the two atoms, preventing the distribution from shrinking and the learning from stopping too early. Finally, the adaptive bounds lead to more accurate estimated action values and better performance.



(a) Fixed bounds (b) Adaptive bounds

Fig. 3. Distribution over return using two methods.

Fig. 2(b) plots the estimated action value and the true action value using the novel Adaptive Bounds (AB) method. From Fig. 2(b), we see that the estimated action value is almost the same as the true action value and the performance of the new method is also better than that of the original CDRL (average value of AB is greater than that of CDRL).

3.3. Distributional target policy smoothing strategy

When applying CDRL to an actor–critic setting, deterministic policies may overfit to narrow peaks in the value estimate [48]. This problem is aggravated when using adaptive bounds, because the new bounds will adapt to different state–action pairs and policies, and so the bounds change with the training process. Every time the bounds change, the expectation of the distribution also changes. There will be a sudden change in the critic network, which will influence the actor network according to the (9): $\nabla_\theta J(\theta) \approx \mathbb{E}_\rho[\nabla_\theta \pi_\theta(x) \mathbb{E}[\nabla_a Z_\omega(x, a)]|_{a=\pi_\theta(x)}]$. This sudden change will increase the variance of the target and influence the accuracy of the estimated action value. Thus, we propose a distributional target policy smoothing method to fix this problem. The proposed method originates from Expected SARSA [20,51]. The update rule of Expected SARSA is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \sum_a \pi(s_{t+1}, a) Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{14}$$

Using the expected value $\mathbb{E}[Q(s_{t+1}, a_{t+1})]$ as the target value, rather than $Q(s_{t+1}, a_{t+1})$, can reduce the variance in the update. Unfortunately, we cannot achieve the same idea in distributional RL and continuous action space problems by enumerating all possible next actions in order to calculate the expected distribution as the target distribution. Thus, we propose the following target distribution:

$$y = r + \mathbb{E}_\varepsilon[Z_{\theta'}(s', \pi_{\phi'}(s') + \varepsilon)] \tag{15}$$

where ε is a small random noise and $\varepsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$. σ determines the range of the noise and the parameter c controls the limit value of the noise. To approximate this expectation, we first sample the noise and add it to the target action. Then, we have a target distribution. This process is repeated K times to give K target distributions. Because these distributions all have the same support set, we can find an approximate expectation by averaging over them. In practice, our proposed DTPS reduces the variance of the update and further corrects the estimated action values.

3.4. Proposed algorithm

In this section, we combine all the components described above and adapt them to an actor–critic setting. The pseudo code of our Adaptive Smoothing-CDRL (AS-CDRL) is presented in Algorithm 1. In step 12, we estimate the confidence interval of the bounds $Q_{max}(s, a)$ and $Q_{min}(s, a)$ and set the upper and lower bound of the intervals as the bounds of our distribution. The procedure of computing Q_{max} can be found in Algorithm 2. Q_{min} is computed in a similar manner. Steps 13–17 describe the DTPS procedure.

Algorithm 1 AS-CDRL

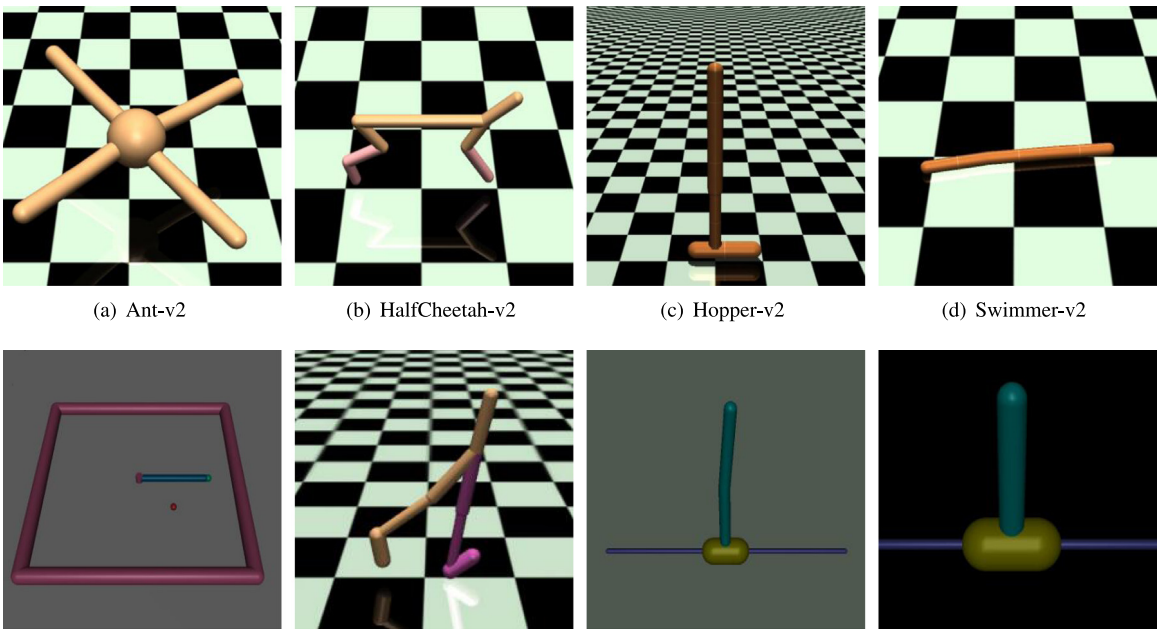
Input:
batch size M, learning rates α and β, sample times K, the frequency of saving the critic network weights N, sample size B

1: Initialize the critic network Q_θ
2: Initialize the actor network π_ϕ
3: Initialize the distributional critic network Z_ω
4: Initialize the replay buffer $\mathscr{B}$
5: Initialize the target weights$(\phi', \omega') \leftarrow (\phi, \omega)$
6: **for** $t = 1$ to T **do**
7: Sample M transitions (s, a, s', r) from $\mathscr{B}$
8: **if** t mod N **then**
9: save critic weights θ_t
10: **end if**
11: sample M transitions (s, a, s', r) from *mathcalB*
12: GetUpperConfidence($\{\theta_1, \ldots, \theta_B\}$,$(s, a)$)
13: **for** $j = 1$ to K **do**
14: $\hat{a} = \pi_{\phi'}(s') + \varepsilon$
15: $z' = r + \gamma Z_{\omega'}(s', \hat{a})$
16: **end for**
17: Compute the target distribution z_{target} through averaging over z'
18: Compute the current distribution $z = Z_\omega(s, a)$
19: Compute the distributional critic loss:
$\delta_\omega = \frac{1}{M} \sum \nabla_\omega D_{KL}(\Phi z_{target} || z)$
20: Compute the actor loss:
$\delta_\phi = \frac{1}{M} \sum \nabla_\phi \pi_\phi(s) \mathbb{E}[\nabla_a Z_\omega(s, a)]|_{a=\pi_\phi(s)}$
21: Update the network parameters:
$\omega \leftarrow \omega + \alpha \delta_\omega$, $\phi \leftarrow \phi + \beta \delta_\phi$
22: Compute the critic loss using (8) and update θ as in other common RL methods.
23: Update target networks
24: **end for**

4. Experiments and discussions

In this section, we describe the experiments conducted to measure the performance of the proposed AS-CDRL on a set of continuous-control tasks. In Section 4.1, we describe the experimental setup and set some important hyperparameters of our method. Section 4.2 evaluates the proposed AS-CDRL and other baselines on some locomotion tasks and presents the corresponding learning curves. Finally, we analyze the effect of AB and DTPS in Section 4.3. Readers wishing to reproduce our method are welcome to use the open-source code.[1]

[1] https://github.com/zhaoyingnan179346/AS-CDRL.



(a) Ant-v2 (b) HalfCheetah-v2 (c) Hopper-v2 (d) Swimmer-v2

(e) Reacher-v2 (f) Walker2d-v2 (g) InvertedDoublePendulum-v2 (h) InvertedPendulum-v2

Fig. 4. Example Mujoco environments.

Algorithm 2 GetUpperConfidence

Input:
last B weights $\{\theta_1, \ldots, \theta_B\}$, block length l, state–action pairs (s, a), confidence level α(=0.05), the number of bootstrapping times K.

Get sample action values:
$Q_B = \{Q(s, a; \theta_1), \ldots, Q(s, a; \theta_B)\}$
Construct the block:
$B_i = (Q(s, a; \theta_i), \ldots, Q(s, a; \theta_{i+l-1}))$
The total number of blocks:$M \leftarrow \lfloor B/l \rfloor$
for $i = 0$ to K **do**
  Sample M blocks and merge them:
  $Q_B^* = (Q^*(s, a; \theta_1), \ldots, Q^*(s, a; \theta_{M*l}))$
  Compute an estimate of the statistic:
  $T_i^* = max(Q_B^*)$
end for
Sort($\{T_1^*, \ldots, T_K^*\}$)
$T_{\alpha/2}^* = (1 - r)T_j^* + rT_{j+1}^*$
$j = \lfloor K\alpha/2 + (\alpha + 2)/6 \rfloor$, $r = K\alpha/2 + (\alpha + 2)/6 - j$
return $2 * max(Q_B) - T_{\alpha/2}^*$

4.1. Setup

Our experiments were conducted using the Mujoco [52] environment interfaced through OpenAI Gym[2] [53]. We chose eight continuous-control locomotion tasks from Mujoco (see Fig. 4).

In CDRL, we need to set the bounds Q_{min} and Q_{max} of the categorical distribution, and these values have an important impact on the performance. Finding the optimal bounds for each task is a process of trial and error. The bounds are listed in Table 1.

For our method, we used the DDPG architecture. Both the actor and the critic contained two-layer feedforward networks of 400 and 300 hidden nodes, respectively. ReLU non linearity was applied between each layer. The critic received both the state and action as input, and produced the atom probabilities as output. Both network parameters were updated using Adam [54]. The learning rate for the actor was $1e^{-4}$ and that for the critic was $2.5e^{-4}$. After each time step, the networks were trained with a mini-batch of 100 transitions.

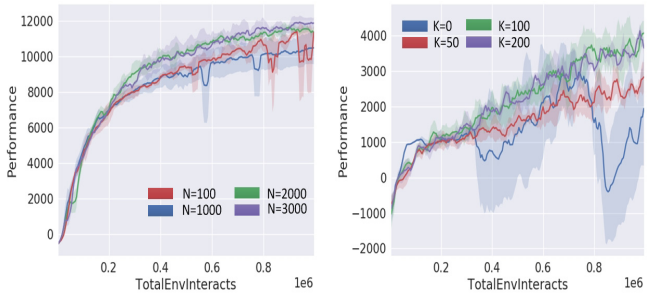
Table 1
Optimal bounds for CDRL in different tasks.

Task	Q_{min}	Q_{max}
Ant-v2	−200	300
HalfCheetah-v2	−100	1100
Reacher-v2	−50	0
Hopper-v2	−60	600
Walker2d-v2	−100	700
Swimmer-v2	0	120
InvertedPendulum-v2	0	160
InvertedDoublePendulum-v2	0	1500

Our proposed AS-CDRL consists of two components: AB and DTPS. For the AB component, we saved the critic weights every N time steps. To provide some intuition, we examined the performance with different values of N on HalfCheetah-v2 (see Fig. 5(a)). With $N = 100$ and $N = 1000$, the training becomes unstable because the frequency of saving the weights also represents the frequency of changing the bounds. If the bounds of the categorical distribution change too often, the estimated action value will also change frequently, which makes the estimated gradients of the actor and the critic inaccurate. Saving the critic weights at a lower frequency ($N = 2000$ and $N = 3000$) improves the performance and makes the training process smoother. This situation is similar in other tasks.

DTPS reduces the variance of the target and stabilizes the training. The sample numbers K in DTPS represent the number of actions we sample in the small area around the target action. We examined the performance with different values of K on Ant-v2 (see Fig. 5(b)). When $K = 0$ (no DTPS), the policy achieves a high score, but the learning process is unstable and causes divergent behavior. As K increases, the approximation of the expected target distribution becomes more accurate and the instability and divergence problems obviously improve. In practice, we set $K = 100$.

[2] https://github.com/openai/mujoco-py.



(a) Ant-v2 (b) HalfCheetah-v2

Fig. 5. Performance of AS-CDRL with different hyperparameters.

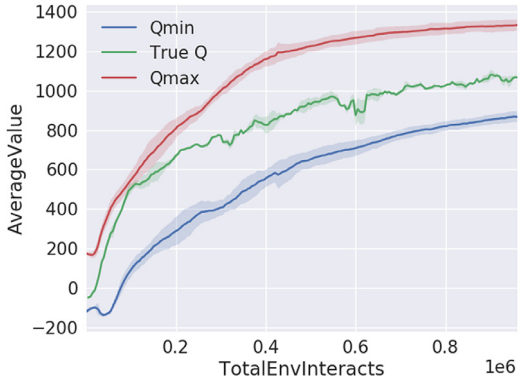


Fig. 6. True averaged action value and the adaptive bounds.

Table 2
Maximum average return over 10 trials of 1 million time steps. The results show the ablation results over DTPS and AB.

Method	HCheetah	Ant	Walker2d	Swimmer
CDRL	10851.26	3295.39	5237.63	122.44
AS-CDRL - AB	11089.11	3476.00	4566.02	140.22
AS-CDRL - DTPS	11542.88	4845.86	5259.77	96.69
AS-CDRL	12217.74	5198.19	5615.16	141.26

The core idea behind AB is that the bounds can be adjusted automatically based on the policy's performance and state–action pairs. To verify this, the bounds and the true action values during the training courses were calculated. From Fig. 6, the values of Q_{min} and Q_{max} increase as the performance (the true action value) improves. The adaptive bounds are not too wide for the true action value, making the estimated action value more accurate.

4.2. Evaluation

We compared our algorithm against DDPG [14], and the state-of-the-art policy gradient algorithms D3PG [33], Twin Delayed Deep Deterministic (TD3) [48], and Proximal Policy Optimization (PPO) [54]. D3PG is an actor–critic version of CDRL that can achieve state-of-the-art performance on many continuous tasks. TD3 reduces the overestimation bias in an actor–critic setting and greatly improves both the learning speed and performance in a number of challenging tasks. These methods like AS-CDRL, are DDPG-family methods that use the *deterministic policy gradient theorem*. PPO is a family of policy optimization methods based on the *stochastic policy gradient theorem*. PPO is motivated by Trust Region Policy Optimization(TRPO) [55] and can take the biggest possible improvement step on a policy. PPO performs well in continuous-control environments. AS-CDRL builds on CDRL by applying AB and DTPS to reduce the estimation error and improve the performance. Besides Categorical distribution, there is also Mixed of Gaussian (MoG) distribution which do not have bounds. We compared our method with the distribution reinforcement learning methods that uses MoG distribution. We denote this method as MoG-KL [33] for using the MoG distribution while minimizing the KL-divergence. MoG-DQN [36] is another distributional reinforcement learning method which used a Mixture of Gaussian distribution to model the distribution of the sum of rewards and proposed a new distance metric for MoG distribution: Jensen–Tsallis Distance (JTD).

Each task was executed over 1 million time-steps. We evaluated the policy for 10 episodes every 5000 time steps and then averaged the rewards. The learning curves are presented in Fig. 7. From the results, we observe that the proposed AS-CDRL achieves better performance than the other methods. AS-CDRL outperforms TD3 on Swimmer-v2, HalfCheetah-v2, Hopper-v2, and InvertedDoublePendulum-v2. AS-CDRL matches or outperforms D3PG in both final performance and learning speed across all tasks because AS-CDRL has a lower estimation error and improved performance compared with D3PG. From Fig. 7 it is clear that MoG-KL does not perform well. Since minimizing a distance metric between two MoG distributions is difficult, neither the KL-divergence nor the Wasserstein distance between two MoG distributions can be computed in a closed form and the estimation of it by a Monte Carlo method is computationally expensive and suffers from high variance. This harms the final performance. MoG-DQN outperforms our method on the environments Swimmer-v2 and Hopper-v2. On other tasks like HalfCheetah-v2, Ant-v2, Walker2d-v2, MoG-DQN underperforms our method due to the lack of constraint on the return distribution, which makes the learned reward distribution inaccurate and brittle. MoG-DQN is completely random at the beginning of the training. Along with the training, the MoG distribution approximates the real rewards distribution gradually, using only the one-step reward. Therefore, the training process is time-consuming and unstable.

To investigate the robustness to random seeds, we record the results for our method and MoG-DQN with different random seeds on all the tasks. We plotted the results on a violin plot in Fig. 8, including markers for the best, worst, and median performances. In Fig. 8, the y-axis represents the normalized score over all the tasks. The white point represents the median score, while the two ends of the line represent the best and worst score. The width of the shape represents the frequency of getting the corresponding score. Fig. 8 shows that the violin plot of our method is wider and shorter, which means that there is little difference between the best and worst score. Furthermore, our method achieve similar results regardless of the different random seeds. The violin plot of MoG-DQN is thinner and longer, which means MoG-DQN may obtain very different performance for just setting different random seeds. This will increase the difficulty of debugging and practical applications. Therefore, our method is much better than MoG-DQN in terms of both the performance and the robustness.

4.3. Ablation

This subsection describes ablations to remove components of our algorithm in order to compare their separate contributions. Our algorithm includes two parts: AB and DTPS. The results presented in Table 2 compare the performance of each component. The significance of each component varies from task to task. Across all four tasks, the best performance is obtained by AS-CDRL. The biggest gain is due to the inclusion of AB, which is particularly helpful on tasks with wide bounds ($Q_{max} - Q_{min}$ is

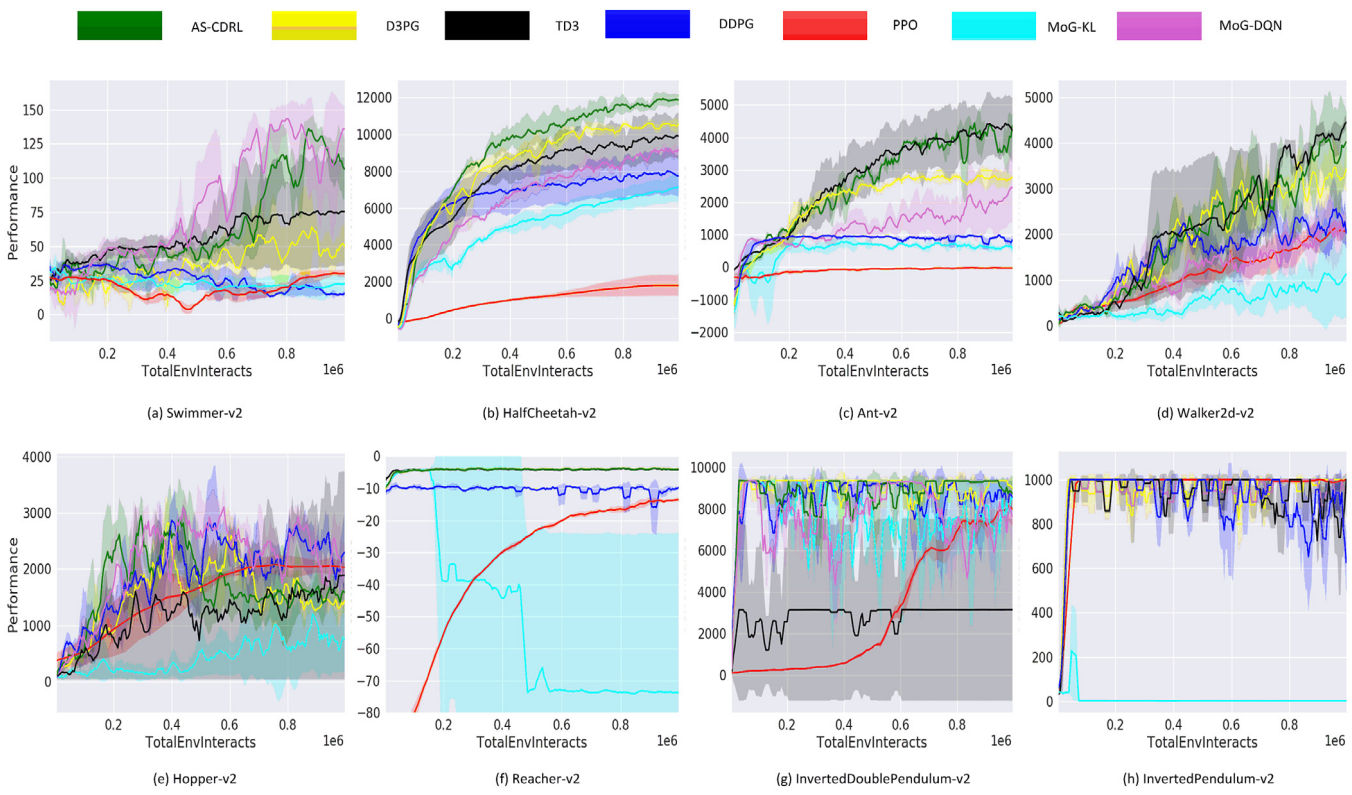


Fig. 7. Learning curves for the OpenAI gym control tasks. The shaded region represents the standard deviation of the average evaluation over three trials.

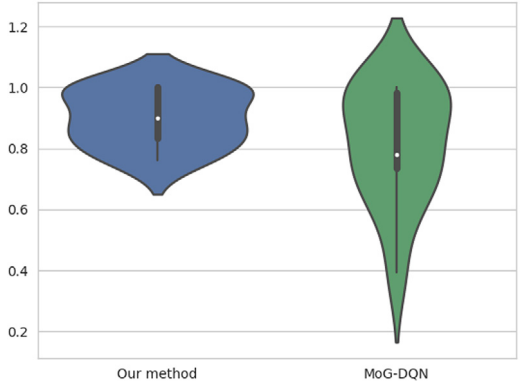


Fig. 8. Distribution of performance across random seeds.

large), which is intuitively reasonable. If $Q_{max} - Q_{min}$ is large, the training produces a greater estimation error and harms the final performance. In tasks like Swimmer-v2, DTPS is more helpful than AB. This is because the range of possible action values is not large and the estimation error is not a serious problem. In such situations, the contribution of AB is not as large as in other tasks.

DTPS can enhance the performance in most of these tasks, especially those with large stochasticity, because the target action for the same state may be different and changes frequently. As a result, the variance of the update is large and increases the bias of the estimated action value. DTPS samples several actions around the target action and smoothes the target distribution. In this way, DTPS can eliminate the effect of stochasticity and improve the performance.

5. Conclusion

This paper has described two novel modifications to CDRL. The first, Adaptive Bounds, saves the critic weights over a fixed number of time steps and obtains the associated action values as the samples for different state–action pairs in different learning stages. *Moving block bootstrapping* then computes confidence intervals for Q_{max} and Q_{min} based on these samples, and the upper and lower bound of these intervals are taken as the adapted bounds.

In practice, AB effectively reduces the estimation error in CDRL. The second modification, Distributional Target Policy Smoothing changes the target distribution to the expected distribution over some similar actions, which further correct the estimated action values.

AB and DTPS constitute our proposed method, Adaptive Smoothing-CDRL. AS-CDRL significantly improves the performance and learning speed of CDRL in eight locomotion tasks taken from Mujoco. Our method outperforms several state-of-the-art algorithms, including TD3 and PPO. In this paper we combine our method with DDPG and show promising avenue for improved performance. Incorporating other policy gradient methods (e.g. trust regions) is an exciting avenue for future work.

CRediT authorship contribution statement

Yingnan Zhao: Writing - original draft, Conceptualization, Methodology, Software. **Peng Liu:** Writing - review & editing, Resources, Methodology, Supervision. **Chenjia Bai:** Validation, Methodology, Writing - review & editing. **Wei Zhao:** Supervision, Writing - review & editing. **Xianglong Tang:** Funding acquisition.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61671175), Science and Technology on Space Intelligent Laboratory, China (ZDSYS-2018-02).

Table 3
Average maximum return for different types of networks.

Method	HCheetah	Ant	Walker2d
FF	12217.7 ± 427.0	5198.1 ± 121.7	4995.6 ± 479.3
RNN	12197.0 ± 102.5	4300.2 ± 901.4	3888.1 ± 358.8
LSTM	11632.0 ± 682.8	3765.2 ± 907.5	4666.6 ± 757.9
GRU	11511.7 ± 241.3	4315.0 ± 475.9	4069.0 ± 276.5

Table 4
Average maximum return for different numbers of hidden node.

Method	HCheetah	Ant	Walker2d
300–200	10834.3 ± 456.3	4587.2 ± 83.9	4183.1 ± 527.7
400–300	12217.7 ± 427.0	5198.1 ± 121.7	4995.6 ± 479.3
400–400	12026.2 ± 430.7	5204.7 ± 285.4	4830.1 ± 370.6
400–500	12390.9 ± 207.9	4802.7 ± 1063.1	4947.1 ± 782.7

Table 5
Average maximum return for different optimizers.

Method	HCheetah	Ant	Walker2d
Adam	12217.7 ± 427.0	5198.1 ± 121.7	4995.6 ± 479.3
Adamax	9785.3 ± 285.3	3338.4 ± 952.3	4663.0 ± 680.0
RMSprop	10669.5 ± 606.8	2476.6 ± 227.5	3308.2 ± 371.7
SGD	7898.3 ± 717.8	1339.3 ± 523.3	2023.2 ± 245.4

Table 6
Average maximum return for different sizes of mini-batch.

Method	HCheetah	Ant	Walker2d
50	10957.2 ± 385.0	4316.5 ± 707.9	4129.2 ± 355.8
100	12217.7 ± 427.0	5198.1 ± 121.7	4995.6 ± 479.3
200	11775.8 ± 616.1	4713.3 ± 211.3	4700.0 ± 707.6
300	12203.7 ± 372.6	5076.7 ± 608.8	5084.2 ± 623.2
400	11740.8 ± 351.6	4861.4 ± 715.0	5061.5 ± 476.4

Appendix A. Networks

In all experiments, we use feedforward networks to parameterize the policy. Actually, there are also some methods that using recurrent networks to solve the problem of partial observability. We add the comparison results with three types of recurrent networks: Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN) and Gated Recurrent Unit (GRU). Table 3 shows the results for three tasks: Ant-v2, Walker2d-v2 and HalfCheetah-v2. Table 3 includes the average maximum return and the standard deviation over five trials.

It is clear that feedforward networks achieve best performance over all three tasks. In our study, the tasks are fully observable, therefore using recurrent networks is unnecessary and increases the complexity of training. In addition, our method is trained using a replay buffer. During training, we sample the transitions from the replay buffer randomly and the relationship between different batches is weak. As a result, the memory stored in recurrent networks is not useful for the training step. So we select the feedforward networks. The full learning curves are shown as Fig. 9.

Appendix B. Number of hidden nodes

The number of the hidden nodes determines the number of the parameters and the expression of the networks. In this paper the policy networks are with 400, 300 hidden units. Here we display the comparison results with other numbers of hidden nodes. Table 4 shows the results for three tasks: Ant-v2, Walker2d-v2 and HalfCheetah-v2. Table 4 includes the average maximum return and the standard deviation over five trials.

From Table 4, the performance of 300 and 200 hidden nodes is the worst. As we increase the number of the hidden nodes, the performance does not improve obviously. 400–300, 400–400 and 400–500 hidden nodes achieve similar results. Considering both the performance and the difficulty of training, we select the 400 and 300 hidden nodes. The full learning curves are shown as Fig. 9.

Appendix C. Optimizer

In this paper, we use Adam as the optimizer for the actor and critic networks. Here we list the comparison results with other three popular optimizers: Adamax, RMSprop and SGD. Table 5 shows the results for three tasks: Ant-v2, Walker2d-v2 and HalfCheetah-v2. Table 5 includes the average maximum return and the standard deviation over five trials.

Table 5 indicates that Adam achieves better performance than three other optimizers. SGD is easy to implement but faces problems when finding a good learning rate and is not suitable for training non-stationary data in reinforcement learning [56]. RMSprop [57] is closely related to Adam; however, there are still a few differences: RMSprop generates its parameter updates using a momentum on the rescaled gradient, whereas Adam updates are directly estimated using a running average of the first and second moment of the gradient. Furthermore, RMSprop lacks a bias-correction term, which leads to very large stepsizes and often divergence [57,58]. Adamax is a variant of Adam based on the infinity norm, which converges to the more stable value but can easily get stuck at the suboptimal solution [57]. Adam combines the advantages of AdaGrad and RMSprop: the ability of AdaGrad to deal with sparse gradients and the ability of RMSProp to deal with non-stationary objectives [57]. Compared with Adamax, Adam is well-suited to a wider range of optimization problems. Therefore we select Adam optimizer. The full learning curves are shown as Fig. 9.

Appendix D. Mini-batch size

In this paper, the mini-batch size we select is 100. The mini-batch size plays an important role in reinforcement learning. If the mini-batch size is large, then the estimated gradient is stable and beneficial to all the transitions. For a small mini-batch size, the estimated gradient is random and only valid for some transitions. Thus, typically, larger mini-batch size leads to better performance.

To justify that our selection of 100 transitions is optimal, we choose five different sizes of mini-batch: 50, 100, 200, 300, 400. Table 6 shows the results for three tasks: Ant-v2, Walker2d-v2 and HalfCheetah-v2. Table 6 includes the average maximum return and the standard deviation over five trials.

Table 6 indicates us that a small batch-size (50) cannot achieve good performance, while 100 transitions achieves good performance over the three tasks. Although, a larger mini-batch size is good for training, the improvement is not obvious as the mini-batch size is increased. This is because increasing the size of mini-batch eliminates the difference between transitions and the gradient descent directions cancel out. Furthermore, larger mini-batch size requires more computing resources. Therefore, we select 100 as our optimal mini-batch size. The full learning curves are shown as Fig. 9.

Appendix E. Learning rate

In this paper, the learning rate we choose is 0.00025. Learning rate is an important hyper parameter for our method. A low learning rate reduces the convergence rate. A large learning rate skips the optimal solution. Here we choose seven different learning rates, ranging from 0.01 to 0.00003. Table 7 shows the results

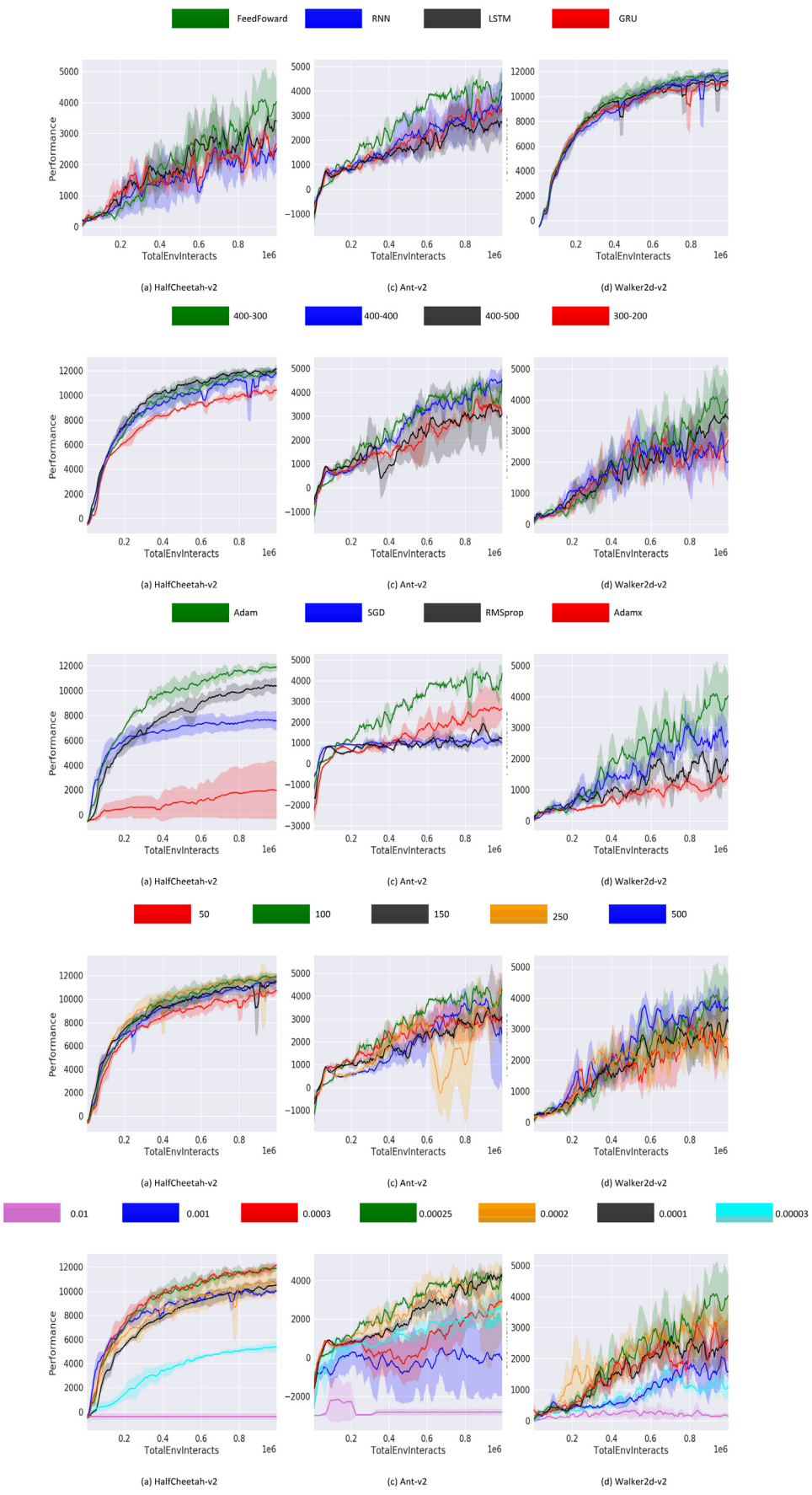


Fig. 9. Learning curves for different hyper parameters.

Table 7
Average maximum return for different learning rates.

Method	HCheetah	Ant	Walker2d
0.01	−289.7 ± 204.4	−1892.3 ± 1250.7	869.3 ± 542.1
0.001	10618.6 ± 218.4	2465.4 ± 273.8	3234.9 ± 637.7
0.0003	12433.2 ± 283.9	3719.6 ± 379.5	4325.2 ± 627.4
0.00025	12217.7 ± 427.0	5198.1 ± 121.7	4995.6 ± 479.3
0.0002	11183.8 ± 1386.1	5047.7 ± 741.5	4692.4 ± 334.6
0.0001	10662.6 ± 648.8	5037.4 ± 201.1	3763.6 ± 458.1
0.00003	5506.9 ± 416.3	3029.1 ± 995.7	2423.5 ± 142.4

Table 8
Hyper parameter values.

Hyper parameter	Value
Discount factor	0.99
Number of hidden nodes	(400, 300)
Activation function	(Relu, Relu, Tanh)
Batch size	100
Learning rate for critic	2.5e−4
Learning rate for actor	1e−4
Noise clip	0.5
Optimizer	Adam
Target network update rate	0.005
Exploration noise	0.1
Evaluation frequency	5000
Replay Buffer size	1e6

for three tasks: Ant-v2, Walker2d-v2 and HalfCheetah-v2. Table 7 includes the average maximum return and the standard deviation over five trials.

As evident, neither a high nor low learning rate can achieve good performance. Considering both the average return and the standard deviation, we select the learning rate of 0.00025. The full learning curves are shown as Fig. 9.

There are also some other critic values, such as the exploration noise, size of the replay buffer, discount factor and activation function. These hyper parameters are always set based on the experimental results and the experience. The exploration noise is set as 0.1. A higher noise causes the agent to explore the environment more but increases the training time, whereas a lower noise causes the agent to get stuck at the suboptimal policy. We set the size of the replay buffer as 1e6. The size of the replay buffer determines the total size of the saved transitions. A smaller size leads to a loss of diversity between the transitions, while a larger size consumes more memory resources. All hyper parameters for our algorithms are shown in Table 8.

References

[1] R.S. Sutton, A.G. Barto, F. Bach, et al., Reinforcement Learning: An Introduction, MIT Press, 1998.

[2] Y. Li, Deep reinforcement learning: An overview, 2017, arXiv:1701.07274.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M.A. Riedmiller, Playing atari with deep reinforcement learning, 2013, arXiv:1312.5602.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M.A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[5] G. Lample, D.S. Chaplot, Playing FPS games with deep reinforcement learning, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4–9, 2017, San Francisco, California, USA, 2017, pp. 2140–2146.

[6] S. Levine, V. Koltun, Guided policy search, in: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16–21 June 2013, 2013, pp. 1–9.

[7] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (2013) 1238–1274.

[8] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, Hindsight experience replay, in: Advances in Neural Information Processing Systems 30: (Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA), 2017, pp. 5055–5065.

[9] C. Bai, P. Liu, W. Zhao, X. Tang, Guided goal generation for hindsight multi-goal reinforcement learning, Neurocomputing (2019).

[10] K. Guu, P. Pasupat, E.Z. Liu, P. Liang, From language to programs: Bridging reinforcement learning and maximum marginal likelihood, in: in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers, 2017, pp. 1051–1062.

[11] Y. Xia, D. He, T. Qin, L. Wang, N. Yu, T.-Y. Liu, W.-Y. Ma, Dual learning for machine translation, in: NIPS, 2016.

[12] W. Luo, P. Sun, F. Zhong, W. Liu, T. Zhang, Y. Wang, End-to-end active object tracking via reinforcement learning, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, StockholmsmäSsan, Stockholm, Sweden, July 10–15, 2018, 2018, pp. 3292–3301.

[13] L. Ren, J. Lu, Z. Wang, Q. Tian, J. Zhou, Collaborative deep reinforcement learning for multi-object tracking, in: Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part III, 2018, pp. 605–621.

[14] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings, 2016.

[15] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016, 2016, pp. 1928–1937.

[16] M.G. Bellemare, W. Dabney, R. Munos, A distributional perspective on reinforcement learning, in: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6–11 August 2017, 2017, pp. 449–458.

[17] B. Radovanov, A. Marcikić, A Comparison of Four Different Block Bootstrap Methods, 2014.

[18] M. White, A.M. White, Interval estimation for reinforcement-learning algorithms in continuous-state domains, in: NIPS, 2010.

[19] B. Efron, Bootstrap methods: Another look at the jackknife, Ann. Statist. 7 (1) (1979) 1–26.

[20] H. van Seijen, H. van Hasselt, S. Whiteson, M.A. Wiering, A theoretical and empirical analysis of Expected Sarsa, in: IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2009, Nashville, TN, USA, March 31 - April 1, 2009, 2009, pp. 177–184.

[21] C. Szepesvári, Algorithms for Reinforcement Learning, in: Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2010.

[22] C.J.C.H. Watkins, P. Dayan, Technical note Q-learning, Mach. Learn. 8 (1992) 279–292.

[23] H.P. van Hasselt, Double Q-learning, in: NIPS, 2010.

[24] G. Tesauro, Temporal difference learning and TD-gammon, ICGA J. 18 (2) (1995) 88.

[25] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999], 1999, pp. 1057–1063.

[26] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 229–256.

[27] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, T. Tanaka, Parametric return density estimation for reinforcement learning, in: UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8–11, 2010, 2010, pp. 368–375.

[28] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, T. Tanaka, Nonparametric return distribution approximation for reinforcement learning, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21–24, 2010, Haifa, Israel, 2010, pp. 799–806.

[29] R. Bellman, Dynamic programming, Science 153 (3731) (1957) 34–7.

[30] M. Rowland, M.G. Bellemare, W. Dabney, R. Munos, Y.W. Teh, An analysis of categorical distributional reinforcement learning, in: AISTATS, 2018.

[31] M. Rowland, R. Dadashi, S. Kumar, R. Munos, M.G. Bellemare, W. Dabney, Statistics and samples in distributional reinforcement learning, 2019, arXiv:abs/1902.08102.

[32] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M.A. Riedmiller, Deterministic policy gradient algorithms, in: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21–26 June 2014, 2014, pp. 387–395.

[33] G. Barth-Maron, M.W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, T.P. Lillicrap, Distributed distributional deterministic policy gradients, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[34] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M.G. Azar, D. Silver, Rainbow: Combining improvements in deep reinforcement learning, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, 2018, pp. 3215–3222.

[35] A. Gruslys, W. Dabney, M.G. Azar, B. Piot, M.G. Bellemare, R. Munos, The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[36] D. Choi, K. Lee, S. Oh, Distributional deep reinforcement learning with a mixture of Gaussians, in: 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 9791–9797.

[37] W. Dabney, M. Rowland, M.G. Bellemare, R. Munos, Distributional reinforcement learning with quantile regression, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, pp. 2892–2901.

[38] P.J. Bickel, D.A. Freedman, Some asymptotic theory for the bootstrap, Ann. Statist. (1981) 1196–1217.

[39] W. Dabney, G. Ostrovski, D. Silver, R. Munos, Implicit quantile networks for distributional reinforcement learning, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, StockholmsmäSsan, Stockholm, Sweden, July 10–15, 2018, 2018, pp. 1104–1113.

[40] C. Lyle, P.S. Castro, M.G. Bellemare, A comparative analysis of expected and distributional reinforcement learning, 2019, CoRR abs/1901.11084, arXiv:1901.11084.

[41] P.S. Castro, S. Moitra, C. Gelada, S. Kumar, M.G. Bellemare, Dopamine: A research framework for deep reinforcement learning, 2018, CoRR abs/1812.06110, arXiv:1812.06110.

[42] Y. Tang, S. Agrawal, Exploration by distributional reinforcement learning, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden, 2018, pp. 2710–2716.

[43] D. Freirich, T. Shimkin, R. Meir, A. Tamar, Distributional multivariate policy evaluation and exploration with the bellman GAN, in: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, 2019, pp. 1983–1992.

[44] V. Tangkaratt, A. Abdolmaleki, M. Sugiyama, Guide actor-critic for continuous control, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[45] B. Dai, A. Shaw, N. He, L. Li, L. Song, Boosting the actor with dual critic, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[46] O. Nachum, M. Norouzi, K. Xu, D. Schuurmans, Trust-PCL: An off-policy trust region method for continuous control, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[47] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, StockholmsmäSsan, Stockholm, Sweden, July 10–15, 2018, 2018, pp. 1856–1865.

[48] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, StockholmsmäSsan, Stockholm, Sweden, July 10–15, 2018, 2018, pp. 1582–1591.

[49] J. Luan, J. Tang, C. Lu, Prediction interval on spacecraft telemetry data based on modified block bootstrap method, in: Artificial Intelligence and Computational Intelligence - International Conference, AICI 2010, Sanya, China, October 23–24, 2010, Proceedings, Part II, 2010, pp. 38–44.

[50] Y. Liu, J. Liang, J. Qian, Moving blocks bootstrap control chart for dependent multivariate data, in: Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: The Hague, Netherlands, 10–13 October 2004, 2004, pp. 5177–5182.

[51] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: A survey, J. Artificial Intelligence Res. 4 (1996) 237–285.

[52] E. Todorov, T. Erez, Y. Tassa, MuJoCo: A physics engine for model-based control, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7–12, 2012, 2012, pp. 5026–5033.

[53] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, 2016, CoRR abs/1606.01540.

[54] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, 2015.

[55] J. Schulman, S. Levine, P. Abbeel, M.I. Jordan, P. Moritz, Trust region policy optimization, in: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6–11 July 2015, 2015, pp. 1889–1897.

[56] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A.Y. Ng, On optimization methods for deep learning, in: ICML, 2011.

[57] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, CoRR abs/1412.6980.

[58] S. Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv:abs/1609.04747.

Yingnan Zhao

Peng Liu

Chenjia Bai

Wei Zhao

Xianglong Tang