Sample-based Distributional Policy Gradient

Rahul Singh, Keuntaek Lee and Yongxin Chen

Abstract— **Distributional reinforcement learning (DRL) is a recent reinforcement learning framework whose success has been supported by various empirical studies. It relies on the idea of replacing the expected return with the return distribution, which captures the intrinsic randomness of the long term rewards. Most of the existing literature on DRL focuses on problems with discrete action space and value based methods. In this work, motivated by applications in control engineering and robotics where the action space is continuous, we propose sample-based distributional policy gradient (SDPG) algorithm. It models the return distribution using samples via a reparameterization technique widely used in generative modeling. We compare SDPG with the state-of-art policy gradient method in DRL, distributed distributional deterministic policy gradients (D4PG). We apply SDPG and D4PG to multiple OpenAI Gym environments and observe that our algorithm shows better sample efficiency as well as higher reward for most tasks[1].**

I. Introduction

As a data-driven counterpart of model-based control, reinforcement learning (RL) has shown potential in solving a variety of complex problems [14], [16]. RL algorithms can be roughly divided into two categories: value function based and policy gradient methods. Value function based algorithms do not explicitly parameterize the policy, but rather obtain the policy from a learned value function. In contrast, policy gradient methods improve a parameterized policy based on the policy gradient theorem [20] and has shown to be more effective in continuous action space control setting. In particular, deep deterministic policy gradient (DDPG) [15] utilizes neural networks to parameterize the policy and have been successful in solving continuous control tasks.

Instead of modeling the value function as the expected sum of the discounted rewards, recently proposed distributional reinforcement learning (DRL) [2] framework suggests to work with the full distribution of random returns, known as value or return distribution. Several typical DRL algorithms such as C51 [2], D4PG [1], QR-DQN [6], and IQN [5] have shown significant performance improvements over non-distributional counterparts in multiple environments including Atari games and DeepMind Control Suite [21]. In DRL, the return distribution is usually represented by discrete categorical form [1], [2], [17] or quantile function [5], [6], [24]. Most of existing work within DRL framework are value function based and thus are not suitable for tasks with continuous action space. One exception is D4PG [1], an actor-critic type policy gradient algorithm based on DRL. It has demonstrated much better performance [1], [21] as compared to its non-distributional counterpart (DDPG). However, it still suffers from various drawbacks such as sample inefficiency and extra burden of parameter tuning, which is largely due to the fact that the return distribution in D4PG is modeled by a discrete categorical distribution.

In this paper, we advocate sample-based representations of return distributions instead of categorical form or quantiles. Our algorithm: sample based distributional policy gradient (SDPG) learns the return distribution by directly generating the return samples via reparameterizing some simple random (e.g. Gaussian) noise samples [13]. SDPG is an actor-critic type policy gradient based algorithm within DRL framework which employs two neural networks: an actor network to parameterize the policy and a critic network to mimic the target return distribution determined via the distributional Bellman equation based on samples. Since the return distribution is usually 1-dimensional for each state-action pair, we leverage the quantile Huber loss as a surrogate of the Wasserstein distance for comparing return distributions and thereby learning the critic network.

Compared to D4PG, SDPG has the following advantages: i) There is no discretization over the value distributions. The value function network is capable of generating any value distributions, which is not categorical as in D4PG; ii) SDPG does not require the knowledge of the range of the return distribution a prior. In contrast, D4PG requires the domain knowledge in terms of bounds on the return distribution; iii) Once the model is trained, the value distribution can be recovered to arbitrary precision by sampling. In contrast, in D4PG, the resolution of the value distribution is fixed once trained. We compare the performance of our algorithm with that of D4PG on multiple OpenAI Gym [4] environments for continuous control tasks. We observe that SDPG exhibits better sample efficiency and performs better than or on-par with D4PG in term of rewards in almost all the environments.

From a control perspective, our method provides a flexible framework to consider more general optimally criteria other than average control cost/reward in policy gradient methods. For instance, one can easily incorporate risk-measures [5], [7], [23] into the picture by passing the return distribution to a risk utility function.

Related Work: Return distribution was first introduced to study Markov decision processes [12], [19]. Recently, it was applied to RL problems [2] which led to DRL. Most of the algorithms proposed under DRL framework are value function based methods that would run into scalability issue for problems with continuous action space. The C51 algorithm [2], a value based algorithm, represented the return distribution using a discrete distribution parameterized by

R. Singh, K. Lee and Y. Chen are with Georgia Tech, Atlanta, GA; emails: rasingh, keuntaek.lee, yongchen@gatech.edu

[1]`https://github.com/rahulsinghchandraul/SDPG`

51 number of uniformly spaced atoms in a specified range. Later, the QR-DQN algorithm [6] proposed to use discrete set of quantiles to represent the return distribution. QR-DQN was further extended in IQN [5] to learn the full quantile function. D4PG [1] and Reactor [9] are the existing policy gradient based methods within DRL framework; D4PG dealt with control problems in continuous action space, whereas Reactor was studied in discrete action settings. However, D4PG also utilized the discrete categorical form to represent the return distribution similar to C51 algorithm [2], which limits the expressive power of the value distribution network.

II. Background

A. Distributional RL

We consider a standard RL problem with underlying model $(\mathcal{X}, \mathcal{A}, R, P, \gamma)$ where $\mathcal{X}, \mathcal{A}$ denote the state and action spaces respectively, $R(x, a)$ is the reward of taking action a at state x, $P(\cdot \mid x, a)$ is the transition kernel and $0 < \gamma < 1$ is the discount factor. The reward R is assumed to be bounded and can be random in general. The state and action spaces could be either discrete or continuous, though we focus on the more challenging continuous setting. The goal of RL is to find a stationary policy π to maximize the accumulated reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)\right]$. Since we focus on continuous-state-action tasks, we restrict the policy π to be deterministic, that is, $a_t = \pi(x_t)$.

The Q-function, denoted by $Q^\pi(x, a)$, describes the expected reward of the agent from taking action $a \in \mathcal{A}$ from state $x \in \mathcal{X}$ at $t = 0$, that is,

$$Q^\pi(x, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)\right],$$
$$x_t \sim P(\cdot \mid x_{t-1}, a_{t-1}),\ a_t = \pi(x_t),\ x_0 = x,\ a_0 = a.$$

It satisfies Bellman's equation [3]

$$Q^\pi(x, a) = \mathbb{E}R(x, a) + \gamma \mathbb{E}Q^\pi(x', \pi(x') \mid x, a). \quad (1)$$

Here we have adopted the convention that x' denotes the state succeeding x, i.e., $x' \sim P(\cdot \mid x, a)$.

In [2], the authors proposed distributional reinforcement learning (DRL), which relies on a random version of Q-function, defined by

$$Z^\pi(x, a) = \sum_{t=0}^{\infty} \gamma^t R(x_t, a_t). \quad (2)$$

Clearly, $Q^\pi(x, a) = \mathbb{E}Z^\pi(x, a)$, namely, Q^π is the statistical mean of the random variable Z^π. So in principle, one should be able to recover Q^π using Z^π. The return distribution Z^π contains extra information such as variance that may be used to incorporate risk in the RL framework. The Z function satisfies a modified Bellman's equation [2]

$$Z^\pi(x, a) = R(x, a) + \gamma Z^\pi(x', \pi(x') \mid x, a), \quad (3)$$

where the equation holds in the probability sense.

Multiple approaches have been proposed to model the return distribution $Z^\pi(x, a)$ including distribution quantiles [6], [24] and discrete categorical distribution [1], [2], [17]. In the latter, $Z^\pi(x, a)$ is described a probability vector/histogram with fixed bins. The positions of the bins are chosen a prior that need tuning according to the environment under consideration.

B. D4PG

The DRL was extended to the policy gradient framework in [1]. In policy gradient framework, the policy π is modeled by a network π_θ directly. For continuous state-action task, a widely used method is deterministic policy gradient, which relies on the deterministic policy gradient theorem [18]. Let $J(\theta)$ be the average return with control strategy π_θ, then

$$\nabla J(\theta) = \mathbb{E}\left[\nabla_\theta \pi_\theta(x) \nabla_a Q^\pi(x, a) \mid_{a=\pi_\theta(x)}\right]. \quad (4)$$

This theorem is generalized to the DRL setting [1], stated as

$$\nabla J(\theta) = \mathbb{E}\left[\nabla_\theta \pi_\theta(x) \mathbb{E}[\nabla_a Z^\pi(x, a)] \mid_{a=\pi_\theta(x)}\right], \quad (5)$$

which follows directly from (4). The distributed distributional deterministic policy gradients (D4PG) [1] algorithm is based on this extension (5). It is an actor-critic type algorithm in which the critic learns the return distribution Z^π via a neural network. Similar to [2], Z is modeled by a categorical discretization for each (x, a) pair. The actor π_θ is updated via the generalized policy gradient theorem (5) with the expectation being replaced by empirical average.

C. Optimal transport and quantile Huber loss

Given two random variable X, Y in the Euclidean space associated with probability distribution μ_X, μ_Y, the Optimal transport (OT) [22] problem seeks the solution to

$$\min_{\nu \in \Pi(\mu_X, \mu_Y)} \int c(x, y) d\nu(x, y) \quad (6)$$

with $\Pi(\mu_X, \mu_Y)$ denoting the set of feasible joint distributions of X and Y. The unit cost function $c(x, y)$ is often taken to be $||x - y||^p$, $1 \le p < \infty$, in which case, (6) defines the Wasserstein distance [22] $W_p(\mu_X, \mu_Y)$ between μ_X and μ_Y.

Computing the Wasserstein distance in general requires solving a linear programming (6), which could be potentially very expensive. One exception is the one-dimensional problem, which has a closed-form solution. When only samples generated by μ_X, μ_Y are available, their Wasserstein distance can be approximated as follows. Let $\{x_1, x_2, \ldots, x_n\}$, $\{y_1, y_2, \ldots, y_n\}$ be i.i.d. samples corresponding to μ_X, μ_Y respectively, and $\{\tilde{x}_i\}$ ($\{\tilde{y}_i\}$) be the ascending sorted version of $\{x_i\}$ ($\{y_i\}$), then

$$W_p(\mu_X, \mu_Y)^p \approx \frac{1}{n} \sum_{i=1}^{n} ||\tilde{x}_i - \tilde{y}_i||^p. \quad (7)$$

However as noted in [6], (7) does not give unbiased approximation of the Wasserstein distance. In general,

$$\operatorname{argmin}_{\mu_X} \mathbb{E}[W_p(\mu_X, \hat{\mu}_Y)] \neq \operatorname{argmin}_{\mu_X} W_p(\mu_X, \mu_Y),$$

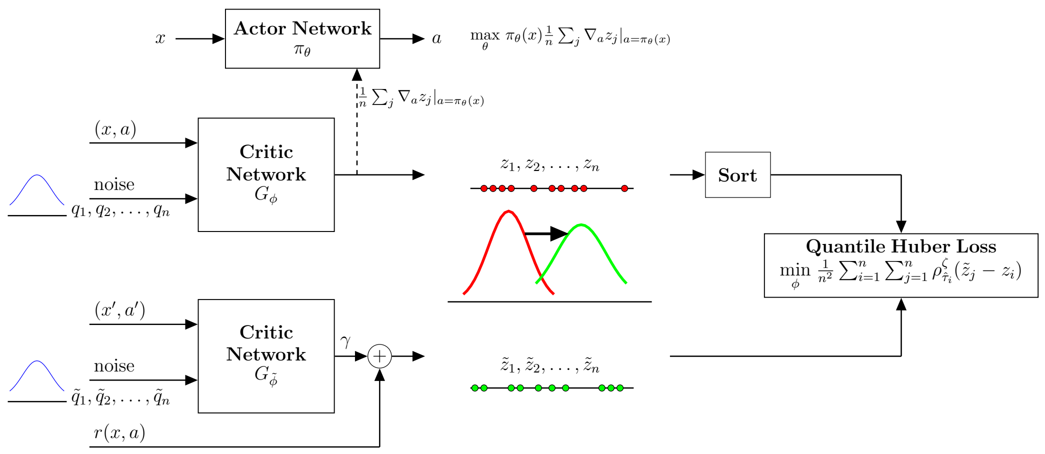

x
Actor Network π_θ
a
$\max_\theta \pi_\theta(x)\frac{1}{n}\sum_j \nabla_a z_j|_{a=\pi_\theta(x)}$
$\frac{1}{n}\sum_j \nabla_a z_j|_{a=\pi_\theta(x)}$
(x, a)
noise
$q_1, q_2, \ldots, q_n$
Critic Network G_ϕ
$z_1, z_2, \ldots, z_n$
Sort
Quantile Huber Loss
$\min_\phi \frac{1}{n^2}\sum_{i=1}^n \sum_{j=1}^n \rho_{\hat{\tau}_i}^{\zeta}(\tilde{z}_j - z_i)$
(x', a')
noise
$\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_n$
Critic Network $G_{\tilde{\phi}}$
γ
$+$
$\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_n$
$r(x, a)$

Fig. 1: Flow diagram of SDPG.

that is, minimizing the distance to the empirical distribution $\hat{\mu}_Y$ composed of samples from one distribution is not equivalent to minimizing the distance to that distribution μ_Y itself [6]. This hinders the use of stochastic gradient methods to minimize the Wasserstein distance. Utilizing the fact that the return distribution is 1-dimensional, one can leverage the quantile Huber loss [6], [11] as an approximation to the Wasserstein-1 distance. The quantile Huber loss acts as a surrogate of the Wasserstein-1 distance, minimizing which effectively minimizes the latter. This is given by

$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\rho_{\hat{\tau}_i}^{\zeta}(\tilde{x}_i - \tilde{y}_i), \tag{8}$$

where $\rho_{\hat{\tau}_i}^{\zeta}(v) = |\hat{\tau}_i - \delta_{\{v<0\}}|L_\zeta(v)$,

$$L_\zeta(v) = \begin{cases} 0.5\, v^2 & \text{if } |v| < \zeta \\ \zeta(|v| - 0.5\,\zeta) & \text{otherwise,} \end{cases}$$

and $\tau_i = \frac{i}{n}, \hat{\tau}_i = \frac{1}{2}(\tau_i + \tau_{i-1}), \quad i = 1, 2, \ldots, n.$

D. Reparameterization

Reparameterization is an effective method to model random variables, especially in case when the goal is to sample from a target distribution instead of modeling them directly using function approximations. Briefly, to model a random variable X with distribution μ_X, reparameterization trick seeks a neural network to map a simple random variable ϵ (e.g. Gaussian) to the target random variable X, that is, $X = G(\epsilon)$. The hope is that after training, the random variable $G(\epsilon)$ has distribution μ_X. This is extremely useful for sampling purpose because one only needs to sample from simple distribution ϵ in order to generate samples of X.

III. Algorithm

Our algorithm SDPG consists of two networks: a critic network to learn the return distribution and an actor network to learn the policy. A flow diagram of the SDPG algorithm is shown in Figure 1. We model the return distribution by samples reparameterized via a simple noise distribution $\mathbb{P}_q$ (we use zero-mean Gaussian distribution with unit variance in our implementation). The critic network G_ϕ, parameterized by ϕ, generates the return samples $z_1, z_2, \ldots, z_n$ for each state and action pair by transforming the noise samples $q_1, q_2, \ldots, q_n$. These generated samples are compared against the samples $\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_n$ generated using the distributional Bellman equation (3) employed in the target critic network $G_{\tilde{\phi}}$. The critic network is learned by minimizing the quantile Huber loss as defined in (8). Thus, the loss function for critic network is

$$L_{critic}(\phi) = \mathbb{E}\left[\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\rho_{\hat{\tau}_i}^{\zeta}(\tilde{z}_j - z_i)\right], \tag{9}$$

where $z_1 \geq z_2 \geq \ldots z_n$ are samples after sorting. We emphasize that the sorting is important here to associate each sample with a reasonable $\hat{\tau}$. This is different to [5] where $\hat{\tau}$ itself is the random seed over $[0\ 1]$.

The actor network π_θ, parameterized by θ, outputs the action $\pi_\theta(x)$ given a state x. The actor network receives feedback from the critic network G_ϕ in terms of the gradients of the return distribution with respect to the actions determined by the policy. This feedback is used to update the actor network by applying distributional form of the policy gradient theorem given by (5). Therefore, the gradient of the actor network loss function is

$$\nabla_\theta L_{actor}(\theta) = \mathbb{E}\left[\nabla_\theta \pi_\theta(x)\ \frac{1}{n}\sum_{j=1}^{n}[\nabla_a z_j]\,|_{a=\pi_\theta(x)}\right] \tag{10}$$

The SDPG algorithm is summarized in Algorithm 1. The network parameters of actor and critic networks are updated alternatively in stochastic gradient ascent/descent fashion.

In contrast to the categorical parameterization of the return distribution considered in D4PG, we use samples to represent the return distribution. Since the return distribution is required to be differentiable with respect to the network parameters in order to be learned, we utilize reparameterization trick discussed in Section II-D to model the return distribution via random noise input. This allows us to learn a continuous distribution via samples as opposed to a discrete-valued

Algorithm 1 SDPG

```
Require: Learning rates $\alpha$ and $\beta$, batch size $M$, sample size $n$, exploration constant $\delta$,
Initialize the actor network $(\pi)$ parameters $\theta$, critic network $(G)$ parameters $\phi$ randomly
Initialize target networks $(\tilde{\theta}, \tilde{\phi}) \leftarrow (\theta, \phi)$
for the number of environment steps do
    Sample $M$ number of transitions $\{(x_t^i, a_t^i, r_t^i, x_{t+1}^i)\}_{i=1}^M$ from the replay pool
    Sample noise $\{q_j^i\}_{j=1}^n \sim \mathbb{P}_q$ and $\{\tilde{q_j}^i\}_{j=1}^n \sim \mathbb{P}_q$, for $i = 1, \dots, M$
    Apply Bellman update to create samples: $\tilde{z}_j^i = r_t^i + \gamma G_{\tilde{\phi}}(\tilde{q_j}^i | (x_{t+1}^i, \pi_{\tilde{\theta}}(x_{t+1}^i)))$ for $j = 1, 2, \dots, n$
    Generate samples $z_j^i = G_\phi(q_j^i | (x_t^i, a_t^i))$ for $j = 1, 2, \dots, n$
    Sort the samples $z^i$ in ascending order
    Update $G_\phi$ by stochastic gradient descent with learning rate $\beta$: $\frac{1}{M}\sum_{k=1}^M \frac{1}{n^2}\sum_{i=1}^n \sum_{j=1}^n \rho_{\hat{\tau}_i}^{\zeta}(\tilde{z}_j^k - z_i^k)$
    Update $\pi_\theta$ by stochastic gradient ascent with learning rate $\alpha$: $\frac{1}{M}\sum_{i=1}^M \pi_\theta(x_t^i) \, \frac{1}{n}\sum_{j=1}^n \left[\nabla_a z_j^i\right]|_{a=\pi_\theta(x_t^i)}$
    Update target networks $(\tilde{\theta}, \tilde{\phi}) \leftarrow (\theta, \phi)$
end for

repeat
    Observe $(x_t, a_t, x_{t+1})$ and draw reward $r_t$
    Sample action $a_{t+1} = \pi_\theta(x_{t+1}) + \delta\mathcal{N}(0, 1)$
    Store $(x_t, a_t, r_t, x_{t+1}, a_{t+1})$ in replay pool
until learner finishes
```

categorical distribution in D4PG. Moreover, D4PG requires a projection step in every iteration during training in order to make the target distribution resulting from the distributional Bellman equation coincide with the support of categorical parameterized distribution being learned, which introduces extra approximation errors; SDPG eliminates the need of such a projection step during training. Furthermore, the range of the discretized grid required in D4PG must be tuned according to the reward values for each environment; SDPG does not require such tuning. Another advantage of SDPG is that one can recover the return distribution to arbitrary precision by sampling from the trained critic network. However, the resolution of the return distribution is fixed in D4PG and the critic network has to be trained again from the scratch if one wants to change the resolution.

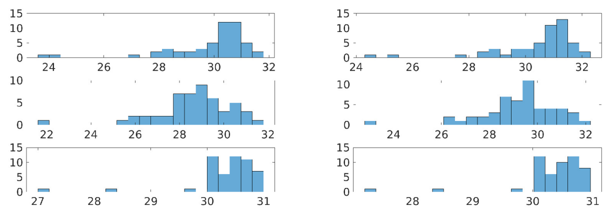


Fig. 2: Histograms of the 51 samples from random state-action pairs after training on BipedalWalker-v2: (left) histograms of samples by the critic network and (right) corresponding histograms with the distributional Bellman equation.

Other than sample based reparametrization, the implicit quantile network (IQN) [5], which learns the reward distribution in terms of cumulative distribution function (CDF) for each state-action pair, can also be extended to policy gradient method for continuous action space. The resulting algorithm is similar to SDPG and is named IQN policy gradient (IQN-PG); details are skipped due to page limit. One potential drawback of the IQN-PG is that the CDFs learned during training are not monotone and thus are not valid CDFs. In our experiments, IQN-PG has comparable performance to SDPG.

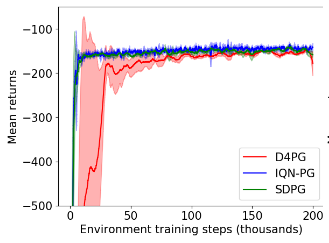
In principle, one can adopt a conditional GAN [8] method to learn the return distribution; this however requires an extra discriminator network. We believe this is an overkill for DRL framework as the return distributions are one dimensional. Our strategy using quantile Huber loss, as a surrogate of the Wasserstein distance, to learn return distributions directly is a better choice. Moreover, we implemented this GAN-based method but couldn't make it work. This might be due to the fact that the state-action pair in our problems takes continuous values instead of discrete values, which makes the training of the conditional GAN unstable.

IV. Experimental Results

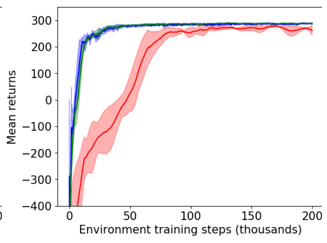
We compare the performance of SDPG with D4PG and IQN-PG algorithms on a range of challenging continuous control tasks from the OpenAI Gym environments. Noise was injected to increase the randomness of the dynamics. Since in [2] and [21], it was demonstrated that D4PG outperforms deep deterministic policy gradient (DDPG) [15] in almost all the environments and therefore, we only compare algorithms within DRL framework.

For both actor and critic networks, we use a two layer feedforward neural network with hidden layer sizes of 400 and 300, respectively, and rectified linear units (ReLU) between each hidden layer. We used batch normalization on all the layers. Moreover, the output of the actor network is passed through a hyperbolic tangent (Tanh) activation unit.

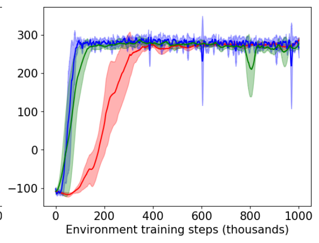
In all experiments we use learning rates of $\alpha = \beta = 1 \times 10^{-4}$, batch size $M = 256$, exploration constant $\delta = 0.3$, and



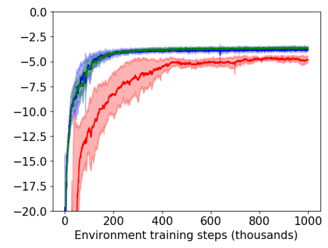
(a) Pendulum-v0.



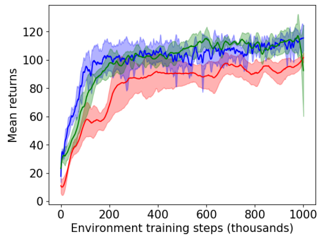
(b) LunarLander-v2.



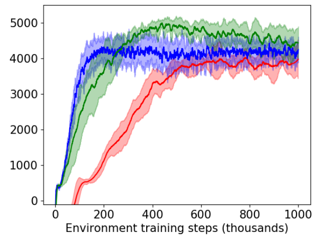
(c) BipedalWalker-v2.



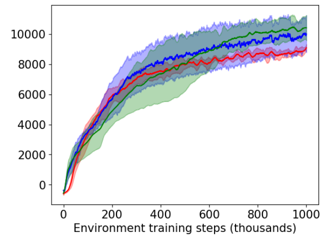
(d) Reacher-v2.



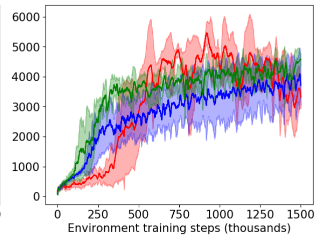
(e) Swimmer-v2.



(f) Ant-v2.



(g) HalfCheetah-v2.



(h) Humanoid-v2.

Fig. 3: Mean returns on different environments. The shaded region represents standard deviation of the average returns over 5 seeds. The curves are smoothed uniformly for visual clarity. Note that in the original D4PG paper, the environments are from DeepMind Control Suite [21] with the rewards bounded in the range $[0, 1]$ for all the domains.

$\zeta = 1$. We use a replay table of size $R = 1 \times 10^6$ for all the domains except $R = 0.2 \times 10^6$ for Pendulum and LunarLander. Across all the tasks, for D4PG we use 51 atoms to represent the categorical distribution and similarly, for SDPG we use $n = 51$ number of samples to represent return distributions. Moreover, we run each task for a maximum of 1000 steps per episode. Note that SDPG is a centralized algorithm at this moment for a single agent. Thus, the distributed feature in D4PG is deactivated for fair comparison. One can easily establish a distributed version of SDPG. Since SDPG requires sorting operation during training, SDPG takes a little more time per episode (roughly $\times 1.3$) than D4PG.

A. *Training and Evaluation*

First we demonstrate the ability of the critic network to learn the return distribution utilizing the quantile Huber loss (8) based on the distributional Bellman equation. Figure 2 shows the histograms of the samples generated by the learned critic network and the corresponding histograms of samples generated based on the distributional Bellman equation on BipedalWalker-v2 domain. Clearly, the histograms match almost perfectly which demonstrates that the critic network in SDPG successfully learns the target return distribution determined via the distributional Bellman equation.

To compare the three algorithms, we train five different instances of each algorithm with random seeds and each performing one evaluation rollout every 5000 environment steps. Figure 3 shows the comparison of mean returns in different environments. It is evident from the figures that SDPG exhibits significantly better sample efficiency than D4PG on almost all the environments. The performance of SDPG is close to IQN-PG when sample efficiency is concerned. Moreover in terms of average returns, SDPG as well as IQN-PG perform better than D4PG on all the domains except Humanoid-v2. This maybe due to insufficient training steps. The performance of SDPG and IQN-PG for Humanoid-v2 keeps increasing during the entire training process and this trend is expected to continue.

We evaluate the performance of the algorithms based on two criteria: average returns and sample efficiency. Table I lists the maximal mean returns (the average of the maximal returns over different trials) along with the standard deviation. The average returns are evaluated every 5000 training steps over 100 episodes. We observe that the returns for SDPG and IQN-PG are significantly better than D4PG for all the environments except Humanoid. To compare the sample efficiency of the three algorithms, we list the number of episodes needed to reach certain return threshold in Table II. The episode numbers reported in the table are averaged over 5 different trials and for each trial the episode number is the number of episodes required before the reward crosses a certain threshold. It is evident that SDPG and IQN-PG require significantly smaller number of episodes than D4PG on many environments.

Lastly, we study the effect of varying number of samples representing the return distribution while training. Figure 4 (a) depicts the training curves with different samples on Ant-v2 domain. For a fixed number of samples, the algorithm is trained for five different seeds: the solid lines represent the mean returns over five trials and the shaded region represent the corresponding standard deviation. Initially, increasing the number of samples improves the performance in terms of efficiency as well as returns, but the trend does not continue for large sample size. Furthermore, Figure 4(b) compares the performance by increasing the number of atoms in D4PG on LunarLander domain. It can be observed that increasing the

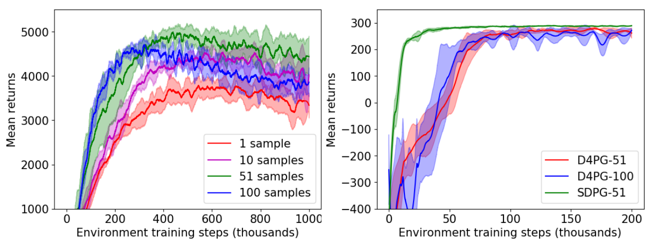
TABLE I: Comparison of average maximal returns $\pm$ one standard deviation over 5 different trials.

			Reward		
Domain	Simulator	Train Steps	D4PG	IQN-PG	SDPG
Pendulum	Classic control	0.2×10^6	-144.63 ± 6.72	$\mathbf{-128.81 \pm 3.35}$	-142.46 ± 5.21
LunarLander	Box2D	0.2×10^6	283.22 ± 1.63	290.68 ± 1.72	$\mathbf{291.71 \pm 0.90}$
BipedalWalker	Box2D	1.0×10^6	289.66 ± 3.26	$\mathbf{302.55 \pm 6.64}$	301.91 ± 4.38
Reacher	MuJoCo	1.0×10^6	-4.40 ± 0.28	$\mathbf{-3.41 \pm 0.15}$	-3.62 ± 0.20
Swimmer	MuJoCo	1.0×10^6	115.52 ± 4.94	126.90 ± 5.91	$\mathbf{128.38 \pm 6.20}$
Ant	MuJoCo	1.0×10^6	5058.84 ± 93.57	4991.20 ± 288.57	$\mathbf{5762.83 \pm 194.67}$
HalfCheetah	MuJoCo	1.0×10^6	9565.06 ± 209.77	10081.30 ± 1362.21	$\mathbf{10607 \pm 845.04}$
Humanoid	MuJoCo	1.5×10^6	$\mathbf{6064.14 \pm 192.84}$	4755.66 ± 623.44	5093.62 ± 402.25

TABLE II: Comparison of sample efficiency in terms of number of episodes needed to reach a certain threshold. The episodes reported here are the smallest m for which the mean episode reward over m^{th} and $(m+10)^{th}$ episodes crosses a certain return threshold. The thresholds are chosen from [10].

		Episodes		
Environment	Threshold	D4PG	IQN-PG	SDPG
Pendulum	-150	1605	1142	**481**
LunarLander	200	2994	1244	**1039**
BipedalWalker	250	4659	**2996**	3039
Reacher	-7	36623	24837	**11859**
Swimmer	90	**3492**	9158	3867
Ant	3500	5567	**4168**	4437
HalfCheetah	4700	**3217**	17585	7692
Humanoid	2500	57142	65915	**51191**

number of atoms in D4PG does not improve the performance.



(a) SDPG on Ant-v2. (b) D4PG on LunarLander-v2.

Fig. 4: Effect of number of samples.

V. Conclusion

In this paper, driven by applications in continuous action space, we proposed sample-based distributional policy gradient (SDPG) algorithm for learning the policy. This algorithm is a combination of an actor-critic type of policy gradient method and DRL. Our algorithm showed better sample efficiency than D4PG in most environments and performed better than D4PG in terms of average returns. We plan to leverage the power of SDPG to cope with more general optimality criteria such as risk.

References

[1] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations*, 2018.
[2] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 449–458, 2017.
[3] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.
[5] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pages 1104–1113, 2018.
[6] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *AAAI Conference on Artificial Intelligence*, 2018.
[7] W. H. Fleming and W. M. McEneaney. Risk-sensitive control on an infinite time horizon. *SIAM Journal on Control and Optimization*, 33(6):1881–1915, 1995.
[8] J. Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014(5):2, 2014.
[9] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, and R. Munos. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. *arXiv preprint arXiv:1704.04651*, 2017.
[10] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *International Conference on Learning Representations*, 2017.
[11] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, pages 73–101, 1964.
[12] S. C. Jaquette. Markov decision processes with a new optimality criterion: Discrete time. *The Annals of Statistics*, pages 496–505, 1973.
[13] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.
[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.
[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
[17] C. Qu, S. Mannor, and H. Xu. Nonlinear distributional gradient temporal-difference learning. *arXiv preprint arXiv:1805.07732*, 2018.
[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.
[19] M. J. Sobel. The variance of discounted Markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.
[20] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.
[21] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. DeepMind control suite. *arXiv preprint arXiv:1801.00690*, 2018.
[22] C. Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.
[23] L. Xia. Optimization of Markov decision processes under the variance criterion. *Automatica*, 73:269–278, 2016.
[24] S. Zhang, B. Mavrin, H. Yao, L. Kong, and B. Liu. Quota: The quantile option architecture for reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2019.