Soft Action Particle Deep Reinforcement Learning for a Continuous Action Space

Minjae Kang*, Kyungjae Lee*, and Songhwai Oh

Abstract— **Recent advances of actor-critic methods in deep reinforcement learning have enabled performing several continuous control problems. However, existing actor-critic algorithms require a large number of parameters to model policy and value functions where it can lead to overfitting issue and is difficult to tune hyperparameter. In this paper, we introduce a new off-policy actor-critic algorithm, which can reduce a significant number of parameters compared to existing actor-critic algorithms without any performance loss. The proposed method replaces the actor network with a set of action particles that employ few parameters. Then, the policy distribution is represented using state action value network with action particles. During the learning phase, to improve the performance of policy distribution, the location of action particles is updated to maximize state action values. To enhance the exploration and stable convergence, we add perturbation to action particles during training. In the experiment, we validate the proposed method in MuJoCo environments and empirically show that our method shows similar or better performance than the state-of-the-art actor-critic method with a smaller number of parameters.**

I. Introduction

Model-free reinforcement learning (RL) with a deep neural network has shown successes to learn complex continuous controllers in many control problems, such as locomotion tasks [1], complex manipulation tasks [2], and autonomous vehicle control [3]. The goal of reinforcement learning is to find a controller, also known as a policy, that best performs a given task. In a general RL problem, since the environment is assumed to be unknown, a robot searches the environment to verify which state and action lead to high performance for a given task through trial and error while maximizing the cumulative rewards at the same time.

Recently, an actor-critic method has been widely used by a neural network that can efficiently learn the policy and value function. In [4], a policy and state action value function (or Q function) is modeled by a neural network where the policy network is updated by the deterministic policy gradient theorem [5] and the value network is updated using the Bellman optimality equation. Moreover, to improve exploration efficiency in continuous action spaces, various methods often utilize entropy maximization [6], [7] for a policy function. In [6], [7], a soft actor-critic method has been proposed which is an actor-critic method to maximize both cumulative rewards and entropy of policy.

There is a drawback of actor-critic methods that require a large number of parameters to model neural networks. First, when computing resources are limited, it is difficult to train a large number of parameters. Accordingly, existing actor-critic algorithms are not suitable for directly learning on a device with small computation capacity such as an embedded computing device. Also, to train many parameters, many hyperparameters should be carefully designed. Furthermore, it is sometimes harmful since a function approximation with large parameters can be easily overfitted where overfitted value and policy functions show poor generalization performance. Thus, the reduction of the number of parameters may improve the generalization performance. Finally, it leads to stable convergence of a policy network and allows us to apply an actor critic method that can tune hyperparameters easily.

In this paper, we propose a new actor-critic method, called a soft action particle (SAP) method, to solve these problems. Unlike other actor-critic algorithms, we reduced the number of parameters by replacing the policy network with action particles created by discretization of the action space. We need to train each action particle as a better particle that is expected to give a higher Q value because we would like to use them to represent the entire action space with only a small number of actions. This method successfully replaces actor networks with a significantly a smaller number of parameters. To measure the performance of SAP, experiments were implemented in MuJoCo simulators. We show the results in *Swimmer* and *HalfCheetah* environments. In conclusion, our proposed algorithm has similar or better performance than the state of the art with fewer parameters than other actor-critic algorithms.

II. Related Work

The actor-critic algorithm has been widely studied in reinforcement learning for a continuous action space. The actor-critic algorithm can be categorized by two main streams: on-policy methods and off-policy methods.

For off-policy methods, deep deterministic policy gradient (DDPG) [4] consists of a deterministic policy network and a Q-network that predicts state-action values. Each network interacts with each other and is trained to obtain higher and more accurate Q values. However, there is a drawback that it does not perform well for complex problems. On the contrary, a soft actor-critic (SAC) [7] method trains a stochastic actor by maximizing both the entropy of policy and the sum of discounted rewards. Since the maximum entropy leads the

*: Equal contribution. M. Kang, K. Lee, and S. Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul 151-744, Korea (e-mail: {minjae.kang, kyungjae.lee, songhwai.oh}@rllab.snu.ac.kr). This work was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01190, (SW Star Lab) Robot Learning: Efficient, Safe, and Socially-Acceptable Machine Learning, and No. 2019-0-01371, Development of Brain-Inspired AI with Human-Like Intelligence).

effective exploration, SAC can achieve stable performance. Twin delayed deep deterministic (TD3) [8] also improves the performance by reducing the overestimation error that is a typical problem of Q-learning. In TD3, clipped double Q-learning method and delayed update of the policy prevent overly optimistic prediction. However, off-policy algorithms use a significant number of parameters because they require various networks for learning.

For on-policy methods, there are various algorithms such as trust region policy optimization (TRPO) [9] and proximal policy optimization (PPO) [10]. TRPO improves the performance of policy slowly by applying the KL-divergence constraint between the old policy and the new policy to prevent a collapse of the policy caused by rapid policy update. PPO suggests a policy training method using only first-order optimization while it keeps the numerical stability of TRPO. PPO penalizes drastic policy change by just clipping the surrogate objective that is the target of maximizing in TRPO. However, these algorithms cannot reuse the past samples gained before. Therefore, whenever a policy is updated, a new sequence of data must be obtained for further training.

Our SAP algorithm is a Q-learning and on-policy method that alters an actor of prior actor-critic algorithms to a set of action particles. By calculating the direction of gradients based on action-state value, action particles are trained to get the higher Q-value. The SAP method shows good performance with fewer amounts of parameters than other actor-critic algorithms. We show the performance of the SAP algorithm compared to another off-policy algorithm DDPG, SAC, and TD3, and on-policy algorithm TRPO and PPO in the MuJoCo environments.

III. Background

In this section, we introduce Markov decision processes which are used as a mathematical framework to formulate a reinforcement learning and entropy regularization of policy distribution. Furthermore, the soft actor-critic method [7] is also reviewed which is the state of the art actor-critic method with the maximum entropy.

An MDP is generally defined as a tuple $\mathbf{M} = \{\mathcal{S}, \mathcal{A}, d, T, \gamma, \mathbf{r}\}$, where $\mathcal{S}$ and $\mathcal{A}$ indicate a state and action spaces, respectively, $d(s)$ is an initial distribution, $T(s'|s,a)$ is a condition probability, or the transition probability from $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ by taking $a \in \mathcal{A}$, $\gamma \in (0,1)$ is a discount factor, and $\mathbf{r}$ is the reward function. The objective of an MDP is to find a policy which maximize $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | \pi, d, T\right]$, where policy π is a mapping from the state space to the action space. For notational simplicity, we denote the expectation of a discounted summation of function $f(s,a)$, i.e., $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t)|\pi, d, T]$, by $\mathbb{E}_{\pi}[f(s,a)]$, where $f(s,a)$ is a function of state and action, such as a reward function $\mathbf{r}(s,a)$ or an indicator function $\mathbf{1}_{\{s=s', a=a'\}}$. We also denote the expectation of a discounted summation of function $f(s,a)$ conditioned on the initial state, i.e., $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t)|\pi, s_0 = s, T]$, by $\mathbb{E}_{\pi}[f(s,a)|s_0 = s]$.

Recently, an MDP is extended to the soft MDP by combining maximum cumulative rewards with maximum entropy objective. It leads to efficient exploration and enables to learn multiple optimal actions since the maximum entropy penalizes the deterministic policy.

A. Soft Markov Decision Processes

In order to obtain a multi-modal policy function and improve sample efficiency, an entropy-regularized MDP, also known as a soft MDP, has been widely used [11]–[14]. A soft MDP is defined as follows:

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \mathbb{E}_{\pi}\left[\mathbf{r}(s,a)\right] + \alpha H(\pi) \\ \text{subject to} \quad & \forall\, s \;\; \sum_{a'} \pi(a'|s) = 1, \;\; \forall\, s, a \;\; \pi(a'|s) \geq 0, \end{aligned} \tag{1}$$

where $H(\pi)$ indicates Shannon-Gibbs entropy regularization, i.e., $\mathbb{E}_{\pi}\left[-\log(\pi(a_t|s_t))\right]$, which is an extension of a general SG entropy to sequential random variables, such as sequence of state and actions, and α is a coefficient. The soft MDP problem (1) has been extensively studied in [6], [12], [13]. In [12], Bloema et al. have derived a soft Bellman equation and the optimal policy distribution from the Karush Kuhn Tucker (KKT) conditions as follows:

$$Q_{\pi}^{soft}(s,a) = \mathbf{r}(s,a) + \gamma \sum_{s'} V_{\pi}^{soft}(s') T(s'|s,a)$$

$$V_{\pi}^{soft}(s) = \alpha \log\left(\sum_{a'} \exp\left(\frac{Q_{\pi}^{soft}(s,a')}{\alpha}\right)\right)$$

$$\pi(a|s) = \frac{\exp\left(\frac{Q_{\pi}^{soft}(s,a)}{\alpha}\right)}{\sum_{a'} \exp\left(\frac{Q_{\pi}^{soft}(s,a')}{\alpha}\right)},$$

$V_{\pi}^{soft}(s)$ is a state value function of π including the entropy of a policy, obtained by starting at state s and $Q_{\pi}^{soft}(s,a)$ is a state-action value function of π with the entropy maximization, which is obtained by starting at state s by taking action a. Note that the optimal policy is a softmax distribution of the state action value function of the soft MDP.

The causal entropy regularization has an effect of making the resulting policy of a soft MDP closer to a uniform distribution as the number of actions increases.

B. Soft Actor Critic Method

In [7], Haarnoja et al. proposed a soft actor critic (SAC) method which can find the optimal policy of the soft MDP. SAC consists of two steps: soft policy evaluation and soft policy improvement. The SAC maintains five networks: policy network π_{ϕ}, two state action value networks Q_{θ_i} for $i = 1, 2$, state value network V_{ψ} and target state value network V_{ψ^-}. In soft policy evaluation step, value networks are updated. For Q_{θ} network, the following loss is used

$$\begin{aligned} L_{\theta} = \mathbb{E}_{(s_t,a_t)\sim M}\Big[\frac{1}{2}(Q_{\theta}(s_t, a_t) \\ - (r(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p_s}[V_{\bar{\psi}}(s_{t+1})]))^2\Big] \end{aligned} \tag{2}$$

which is derived from the soft Bellman equation. The loss function for V_ψ is used as follows:

$$L_\psi = \mathbb{E}_{s_t\sim M}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t|s_t)])^2\right], \quad (3)$$

which is also derived by the relationship between Q and V. In soft policy improvement step, policy function is updated to maximizes the value function which is computed at soft policy evaluation. Since the optimality condition of a soft MDP is that the policy distribution are exponentially proportional to the state action value, policy function is updated to minimize Kullback-Leibler (KL) divergence between policy distribution and softmax distribution of state action value defined as follows:

$$L_\phi = \mathbb{E}_{s_t\sim M}\left[D_{KL}\left(\pi_\phi(\cdot|s_t)\middle\|\frac{\exp(Q_\theta(s_t,\cdot))}{Z_\theta(s_t)}\right)\right]. \quad (4)$$

In [7], it was shown that iteratively update policy and value functions converges to the optimal policy of soft MDP. Furthermore, it shows the state-of-the-art performance compared to prior on-policy and off-policy methods, experimentally.

Since SAC requires five neural networks, optimizing hyperparameters to obtain the best performance requires demanding validation tasks. To handle this issue, the proposed method reduces the number of function approximations in the actor-critic method by replacing the policy network with action particles.

IV. Proposed Method

We propose an action particle method by discretizing the continuous action space into a set of finite actions. The action space is partitioned by N_A number of Gaussian distributions with mean and standard deviation parameters as follows:

$$M = \{\mu_i\}_{i=1}^{N_A},\ S = \{\sigma_i\}_{i=1}^{N_A},$$

where M is a set of means of actions, S is a set of standard deviations. In training step, the agent always samples new action particles as follows:

$$a_i = \mu_i + \epsilon_i\sigma_i,\ \ \epsilon_i \sim \mathcal{N}(0,1),$$

where $a_i \in \mathcal{A}$ is the ith action particle generated by Gaussian distribution. Sampling action set encourages exploration.

Based on N_A actions, a policy distribution is defined by a softmax distribution over N_A actions, which assigns the probability exponentially proportional to $Q_\theta(s,a)$ as follows,

$$\pi_\theta(a_i|s) \triangleq \frac{\mathbb{E}_{\epsilon_i}\left[\exp\left(Q_\theta(s,a_i)/\alpha\right)\right]}{\sum_{m_j\in M,\sigma_j\in S}\mathbb{E}_{\epsilon_j}\left[\exp\left(Q_\theta(s,a_j)/\alpha\right)\right]}, \quad (5)$$

where ϵ_i is a Gaussian noise for the ith Gaussian distribution, and α is an entropy regularization coefficient. The meaning of the expectation over ϵ_i is that we consider all possible actions, which can be sampled from the ith Gaussian distribution. However, in the implementation, the expectation will be approximated by a Monte Carlo sampling method.

A. Soft Q Learning with Action Particles

We update the Q function based on the soft Bellman optimality equation. While the integral over action space is often intractable, we approximate it using the summation over the action particles. This approximation can be viewed as Monte-Carlo integration. The target value of the Q function is computed as follows:

$$y_t = r_t + \gamma\alpha\log\left(\int_{\mathcal{A}}\exp\left(\frac{Q_{\theta^-}(s_{t+1},a)}{\alpha}\right)d\mathbf{a}\right)$$
$$\approx r_t + \gamma\alpha\log\left(\sum_{\mu_j,\sigma_j}\mathop{\mathbb{E}}_{\epsilon_j}\left[\exp\left(\frac{Q_{\theta^-}(s_{t+1},a_j)}{\alpha}\right)\right]\right),$$

where θ^- indicates the parameter of a target network. Since the expectation with respect to ϵ is often intractable, we approximate it using a Monte Carlo sampling method in its implementation where one sample estimation is enough to make the state action value network converge stably.

Then, the Q function can be trained by minimizing the following loss: $L = \frac{1}{B}\sum_{t=0}^{B}\left(y_t - Q_\theta(s_t,a_t)\right)^2$, where θ is the parameter of a prediction network and B is the number of state, action and target pairs. Similarly to other deep reinforcement learning algorithms, we utilizes two networks with parameters θ^- and θ. θ^- is used to estimate y_t and is updated slowly by mixing θ with a ratio τ.

B. Action Particle Update

The performance of the implicit policy π_θ with action particles can be improved by updating Q function towards the optimal action value using the soft Bellman equation. However, it has the limitation in that the fixed particles possibly cannot represent the optimal policy since the optimal actions may not be included in the fixed M. Therefore, we need to move the mean and standard deviation of action particles towards the maximum action value based on a Q function.

Before discussing how to update the particles, we would like to restate the objective function of soft RL. The expected sum of rewards with entropy can be represented as the expectation of the value function with respect to an initial state as follows: $\mathbb{E}_\pi\left[\mathbf{r}(s_t,a_t)\right]+\alpha H(\pi) = \mathbb{E}_{s_0\sim d(s)}\left[V_\pi(s_0)\right]$. Then, we follow the prior work of an off-policy actor-critic method [15]. We want to change the initial state distribution from $d(s)$ to $\rho_\beta(s)$ which is a stationary distribution over the state space when following an arbitrary behavior policy β. The optimal solution with the changed initial distribution is the same as that of the original problem since the soft Bellman optimality equation does not include the initial state distribution. In other words, since the Bellman optimality equation induced by $\mathbb{E}_{s_0\sim d(s)}\left[V_\pi(s_0)\right]$ is exactly equivalent to the optimality with $\mathbb{E}_{s_0\sim\rho_\beta(s)}\left[V_\pi(s_0)\right]$, the optimal solution is not changed. Now, we can freely utilize the off-policy manner by setting ρ_β to be the replay buffer $\mathcal{D}$.

Hence, we aim to update the action particles A to maximize $\mathbb{E}_{s_0\sim\mathcal{D}}\left[V_\pi(s_0)\right]$. Similarly to other Q learning methods, we approximate the state value V_π using the Q_θ function as follows:

$$\begin{aligned} V_\pi(s) &\approx \mathbb{E}_{a\sim\pi_\theta,\{\epsilon\}}\left[Q_\theta(s,a) - \alpha\log(\pi_\theta(a|s))\right] \\ &= \mathbb{E}_{a\sim\pi_\theta}\left[\alpha\log\left(\sum_{a_j\in A}\mathbb{E}_{\mu_j,\sigma_j}\left[\exp\left(\frac{Q_\theta(s,a_j)}{\alpha}\right)\right]\right)\right] \\ &= \alpha\log\left(\sum_{\mu_j,\sigma_j}\mathbb{E}_{\epsilon_j}\left[\exp\left(\frac{Q_\theta(s,a_j)}{\alpha}\right)\right]\right), \end{aligned} \tag{6}$$

where π_θ is replaced with the equation (5). The gradient of the expected return with $\mathcal{D}$ gives proper directions of updating the means and standard deviations, which can be computed as follows:

$$\begin{aligned} &\nabla_{\mu_i}\mathbb{E}_{s\sim\mathcal{D}}\left[V_\pi(s)\right] \\ &= \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_{\mu_i}\alpha\log\left(\sum_{\mu_j,\sigma_j}\mathbb{E}_{\epsilon_j}\left[\exp\left(\frac{Q_\theta(s,a_j)}{\alpha}\right)\right]\right)\right] \\ &= \mathbb{E}_{s\sim\mathcal{D}}\left[\frac{\mathbb{E}_{\epsilon_i}\left[\exp\left(Q(s,a_i)/\alpha\right)\nabla_a Q_\theta(s,a)|_{a=a_i}\right]}{\sum_{\mu_j,\sigma_j}\mathbb{E}_{\epsilon_j}\left[\exp\left(Q(s,a_j)/\alpha\right)\right]}\right], \end{aligned} \tag{7}$$

and

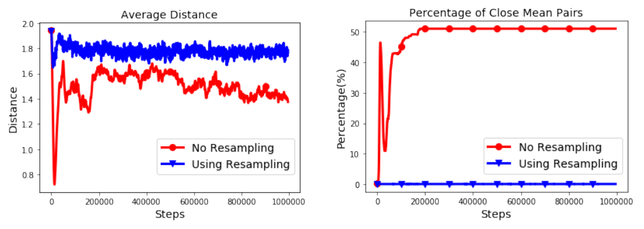
$$\begin{aligned} &\nabla_{\sigma_i}\mathbb{E}_{s\sim\mathcal{D}}\left[V_\pi(s)\right] \\ &= \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_{\sigma_i}\alpha\log\left(\sum_{\mu_j,\sigma_j}\mathbb{E}_{\epsilon_j}\left[\exp\left(\frac{Q_\theta(s,a_j)}{\alpha}\right)\right]\right)\right] \\ &= \mathbb{E}_{s\sim\mathcal{D}}\left[\frac{\mathbb{E}_{\epsilon_i}\left[\exp\left(Q(s,a_i)/\alpha\right)\nabla_a Q_\theta(s,a)|_{a=a_i}\epsilon_i\right]}{\sum_{\mu_j,\sigma_j}\mathbb{E}_{\epsilon_j}\left[\exp\left(Q(s,a_j)/\alpha\right)\right]}\right], \end{aligned} \tag{8}$$

where ∇_{μ_i} is a gradient with respect to the ith mean, ∇_{σ_i} is a gradient with respect to the ith standard deviation, a_i is an action particle that equal to $\mu_i+\epsilon_i\sigma_i$, and $\mathcal{D}$ is a replay buffer. From the equation (7), the moving direction of means is the weighted sum of gradient of the estimated action value. It can be observed that $\nabla_a Q_\theta(s,a)|_{a=a_i}$ indicates the direction of maximizing the expected return and the $\exp\left(Q(s,a_i)/\alpha\right)/Z$ plays a role of an importance weight of action a at state s where Z is $\mathbb{E}_{\{\epsilon\}}\left[\sum_{a_j\in A}\exp\left(Q(s,a_j)/\alpha\right)\right]$. In other words, If $\exp\left(Q(s,a_i)/\alpha\right)$ has high probability, then, it means a_i is a crucial action for state s. The weight makes a_i to maximize $Q_\theta(s,a)$ by amplifying the gradient of $Q_\theta(s,a)$. Since the expectation with respect to ϵ is intractable in practice, we approximate the gradients using Monte Carlo sampling method in the implementation and verify that one sample approximation is sufficient to train the action particles.

C. *Bounded Action Space*

In many robotics problems, the action range of an agent is limited by several conditions. We assume that the given action spaces are bounded by l_a in all dimensions where, for all $a\in\mathcal{A}$, $|a|_{\max}\le l_a$ holds where $|\cdot|_{\max}$ indicates an infinite norm.

During the update of action particles, some particles could violate the bounds of the action space. In that case, it is required to adjust the scale of the violated action values. To maintain the proportion of action particle, the mean vector is divided by a particular value. In brief, if the mean particle μ_i exceeds a boundary value l_a, μ_i is replaced with the projected mean μ_i^+ defined as $\mu_i^+ = \frac{\mu_i}{|\mu_i|_{\max}}l_a$.



(a) Average Distance of Mean Pairs (b) Percentage of Close Mean Pairs

Fig. 1: This graph shows results of the convergence of action means depending on resampling. (a) The change of average distance between all mean pairs when we use resampling or not. (b) The percentage of mean pairs which is closer than certain distance when we use resampling or not.

D. *Resampling*

While updating particle with the weighted gradient can improve the performance of π_θ, it may cause that several particles converge into the same point. This phenomenon can hamper the exploration of the proposed method and causes memory inefficiency.

The red line in Figure 1 shows this problem clearly. As the training of particles progressed, the average distance between mean pairs becomes smaller and the ratio of pairs who closer than particular distance increases. In this regards, to spread the convergent particles, a simple resampling technique is used. We calculate the Euclidean distance between two mean particles and check the following inequality:

$$||\mu_i - \mu_j||_2 < d_{\min} \tag{9}$$

where $d_{\min}$ is the predefined minimum distance. If the inequality (9) is satisfied, we delete a_j and a new action is sampled from uniform distribution in the action space, $\mu\sim\text{Uniform}\left(\mathcal{A}\right)$. The blue line in Figure 1 shows the result of applying the resampling rule. It means that resampling step solves the convergence problem properly.

V. Experiments

A. *Ablation Study*

In this experiment, we conduct several ablation studies to verify the influence of each method to update action particles. We analyze three techniques: updating particles with a naive gradient $\mathbb{E}_{s\sim B}\left[\nabla_a Q_\theta(s,a)\right]$, updating particles with a weighted gradient $\mathbb{E}_{s\sim B}\left[\pi_\theta(a|s)\nabla_a Q_\theta(s,a)\right]$, a re-sampling method. We compare the following methods which are the combinations of three techniques on *HalfCheetah* environment.

1) Fixed Particles: This method is the baseline which initially samples actions from the uniform distribution over the action space and fixes the sampled particles. Only Q_θ is updated.

2) Gradient Update (Grad): This method updates both Q network and particles with a naive gradient $\mathbb{E}_{s\sim B}\left[\nabla_a Q_\theta(s,a)\right]$.

3) Gradient Update with Resampling (Grad+Res): This method updates both Q network and particles with a naive gradient $\mathbb{E}_{s\sim B}\left[\nabla_a Q_\theta(s,a)\right]$ and resamples the particles by checking the condition (9).

Algorithm 1 Soft Action Particle

```
Input: Environment $env$, action space $\mathcal{A}$, the minimum distance $d_{\min}$, the number of action particles $N_A$, update ratio $\tau$
 1: Initialize mean particles $M = \{\mu_i\}$ uniformly sampled from $\mathcal{A}$, standard deviation of particles $S = \{\sigma_i\}$ uniformly sampled from $(0, \sigma_{\max})^D$, replay memory $\mathcal{D} = \emptyset$, Q network parameters $\theta$ and $\theta^-$
 2: for $i = 0$ to $N$ do
 3:     Sample initial state $s_0 \sim d_0(s)$
 4:     for $t = 0$ to $T$ do
 5:         Generate action set $A = \{a_i\}$ using $\mu$ and $\sigma$ particles
 6:         Sample action $a_t \sim \pi_\theta(a|s_t)$ (5)
 7:         Execute $a_t$ and observe $s_{t+1}$ and $\mathbf{r}_t$ from $env$
 8:         Add experiences to replay memory, $\mathcal{D} \leftarrow (s_t, a_t, \mathbf{r}_t, s_{t+1}) \cup \mathcal{D}$
 9:         Sample mini-batch $B$ from $\mathcal{D}$
10:         Set a target value $y_j$ of $(s_j, a_j, \mathbf{r}_j, s_{j+1})$ in $B$, $y_j = \mathbf{r}_j + \gamma\alpha \log\left(\sum_{a_i \in A} \exp\left(\frac{Q_{\theta^-}(s_{j+1}, a_i)}{\alpha}\right)\right)$
11:         Minimize $\sum_j (y_j - Q_\theta(s_j, a_j))^2$
12:         Update $\theta^- \leftarrow (1-\tau)\theta^- + \tau\theta$
13:         if Every $c$ steps then
14:             Update mean and standard deviation particles $\{\mu_j\}$ and $\{\sigma_j\}$ with (7) and (8)
15:             for Every $\mu_j$ and $\mu_k$ pairs in $M$ do
16:                 if $|\mu_j - \mu_k|^2 < d_{\min}$ then $M \leftarrow (M - \{\mu_k\}) \cup \{\mu_{new}\}$ where $\mu_{new} \sim$ Uniform$(\mathcal{A})$
```

4) Soft Action Particle (SAP): The Soft Action Particle (SAP) method is the proposed method which updates both Q network and particles with a weighted gradient $\mathbb{E}_{s\sim B}[\pi_\theta(a|s)\nabla_a Q_\theta(s,a)]$ and resamples the particles by checking the condition (9).

The above methods use the identical architecture for Q network and all hyperparameter settings are the same. 128 action particles are used, i.e., $N_A = 128$ and particles are trained every 100 steps. The minimum distance that is the threshold of resampling is 0.3, and the update ratio of parameters τ is 0.001. We run all algorithms with five different random seeds and measure the average returns during training phase.

The results are shown in Figure 2. By comparing the results of Fixed Particles and Grad, updating particles with a naive gradient leads to a catastrophic failure. This result is mainly caused by the convergence of particles where most particles are concentrated into the same vector. This problem can be alleviated by the resampling method. Grad+Res method shows a reasonable performance similar to the Fixed Particles but still worse. The gradient $\nabla_a Q_\theta(s,a)$ provides the direction to maximizes the average returns at given state s. However, the expected gradient over states smooths each $\nabla_a Q_\theta(s,a)$. In other words, if two different state s and s' generate the conflict gradient directions, then, $\mathbb{E}_{s\sim B}[\nabla_a Q_\theta(s,a)]$ gives not suitable direction and it may causes performance drop as shown in Figure 2.

To handle this issue, the weighted gradient derived from (7), i.e., $\pi(a|s)\nabla_a Q_\theta(s,a)$ is applied. $\pi(a|s)$ can be interpreted as the level of importance a at state s. The high probability of $\pi(a|s)$ means that $Q_\theta(s,a)$ is higher than other actions. To this end, weighted gradient with resampling which is the proposed method outperforms other compared methods.

B. Continuous Control Problems

In this experiment, we compare the proposed method to DDPG, SAC, TD3, TRPO and PPO with the generalized advantage estimation (GAE). Both actor and value networks of TRPO and PPO are implemented with two hidden layers each of that is size 64. Also, all actors, critics, and value networks used in DDPG, TD3, and SAC are composed of two hidden layers, 400, 300 in order. On the other hand, the critic network of SAP is implemented in three layers with sizes of 400, 300, and 300 in order.

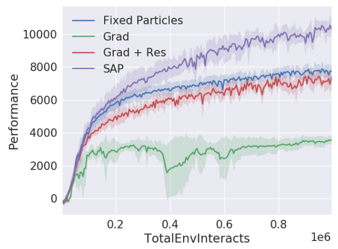
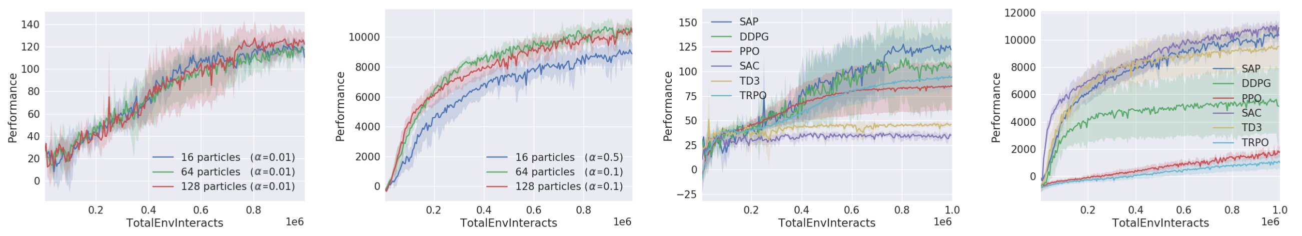


Fig. 2: Average returns from five different random seeds in *HalfCheetah*. Shaded regions indicate the standard deviation.

In Figure 3(a) and (b), the SAP method with different number of particles N_A is tested on the *Swimmer* and *HalfCheetah*. We would like to note that the coefficient of entropy of each method is found by a brute force search. The best coefficients for each number of particles are reported in the legend of Figure **??**(a). As we expect, when more particles are used, SAP achieves better performance. In detail, *Swimmer* is an environment that has two dimensional actions, so the performance with only 16 action particles is not significantly poor compared to the performance with 64 or 128 particles. Rather fewer actions lead to faster learning speed because exploration is successfully carried out to entire action set. But stability is lower than more particle cases. On the other hand, in *HalfCheetah* which has more complex actions, the performance with 64 or 128 particles is much better than 16 particles. This means that 16 actions are not sufficient to cover the entire action space. These results prove efficiency of SAP compare to other discretization algorithms like DQN which requires thousands of actions but hold a



(a) SAP results on Swimmer-v2 with the different number of Particles

(b) SAP results on HalfCheetah-v2 with the different number of Particles

(c) Results on Swimmer-v2 with various algorithms

(d) Results on HalfCheetah-v2 with various algorithms

Fig. 3: Experimental results on the Swimmer and HalfCheetah problems. (a, b) Average returns of SAP with the different number of action particles $16, 64$, and 128. α is the best regularization coefficient in each case. (c, d) Average returns of various actor-critic algorithms. We compared SAP with on-policy methods such as TRPO and PPO, and off-policy methods DDPG, TD3, and SAC. All graphs indicate the average of 10 episodes at every 5000 steps with five different random seeds.

poor performance.

Table I shows how many parameters each algorithm requires in the *HalfCheetah* environment. In the case of SAP, 128 action particles are used as a reference. On the other hand, TRPO and PPO use a large network and a small network for performance experiments and the result of a small network is better for all environments. However, even better performance is much lower than SAP, so it is not meaningful to compare the number of parameters. Therefore, only the actor-critic algorithms are described in the Table I.

Figure 3(c) and (d) shows that SAP performs the highest performance with least deviation in *Swimmer* and indicates the similar performance to state-of-the-art in *HalfCheetah* using the least number of particles. In the case of *HalfCheetah*, it can be seen from Table I that the SAP method uses only 42.7% of total parameters compared to SAC, which is the best performing algorithm. It means action particles that consist of very few parameters can replace the actor that is made up of tens of thousands of parameters in other actor-critic methods.

Algorithm	π	Q	V	total
SAP	1536	220201	-	221737
DDPG [4]	129306	130201	-	259507
TD3 [8]	129306	260402	-	389708
SAC [7]	131112	260402	127801	519315

TABLE I: The number of parameters used in HalfCheetah

VI. Conclusion

In this paper, we have proposed a new actor-critic method, SAP. The main idea of SAP is to replace an actor of existing actor-critic algorithms with a set of action particles. This means that SAP solves robotics problems with a random discretization of action space. Action particles are made by means set and standard deviations set, and these two sets are subject to training. The convergence problem that occurs when we train particles is resolved by resampling technique. Also, we increase the performance of SAP by changing the gradients applied to each particle from a simple sum to a weighted sum. Ablation experiments in the *HalfCheetah* problem have clearly demonstrated its effectiveness. In the experiments, we confirmed that the performance of SAP is higher than or similar to the existing actor-critic algorithms. In other words, by successfully training the action particles, we can effectively replace the actor of existing methods with very few parameters.

References

[1] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics, TOG*, vol. 36, no. 4, p. 41, 2017.

[2] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *IEEE International Conference on Robotics and Automation, ICRA*, May 2015, pp. 156–163.

[3] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation, ICRA*, May 2017, pp. 3357–3364.

[4] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[5] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31th International Conference on Machine Learning, ICML*, Jun 2014, pp. 387–395.

[6] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. of the 34th International Conference on Machine Learning*, Aug 2017, pp. 1352–1361.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and 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*, Jul 2018, pp. 1856–1865.

[8] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning, ICML*, Jul 2018, pp. 1582–1591.

[9] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32th International Conference on Machine Learning, ICML*, Jul 2015, pp. 1889–1897.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, 2017.

[11] M. Tokic and G. Palm, "Value-difference based exploration: adaptive control between epsilon-greedy and softmax," in *Annual Conference on Artificial Intelligence*, Oct 2011, pp. 335–346.

[12] M. Bloem and N. Bambos, "Infinite time horizon maximum causal entropy inverse reinforcement learning," in *Proc. of the IEEE Conference on Decision and Control*, Dec 2014, pp. 4911–4916.

[13] J. Schulman, P. Abbeel, and X. Chen, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

[14] P. Vamplew, R. Dazeley, and C. Foale, "Softmax exploration strategies for multiobjective reinforcement learning," *Neurocomputing*, vol. 263, pp. 74–86, Jun 2017.

[15] T. Degris, M. White, and R. S. Sutton, "Linear off-policy actor-critic," in *Proceedings of the 29th International Conference on Machine Learning, ICML*, 2012.