ADAPTIVE DISCRETIZATION FOR CONTINUOUS CONTROL USING PARTICLE FILTERING POLICY NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Controlling the movements of highly articulated agents and robots has been a long-standing challenge to model-free deep reinforcement learning. In this paper, we propose a simple, yet general, framework for improving the performance of policy gradient algorithms by discretizing the continuous action space. Instead of using a fixed set of predetermined atomic actions, we exploit particle filtering to adaptively discretize actions during training and track the posterior policy distribution represented as a mixture of Gaussians. The resulting policy can replace the original continuous policy of any given policy gradient algorithm without changing its underlying model architecture. We demonstrate the applicability of our approach to state-of-the-art on-policy and off-policy baselines in challenging control tasks. Baselines using our particle-based policies achieve better final performance and speed of convergence as compared to corresponding continuous implementations and implementations that rely on fixed discretization schemes.

1 INTRODUCTION

In the last few years, impressive results have been obtained by deep reinforcement learning (DRL) both on physical and simulated articulated agents for a wide range of motor tasks that involve learning controls in high-dimensional continuous action spaces (Lillicrap et al., 2015; Levine et al., 2016; Heess et al., 2017; Haarnoja et al., 2018c; Rajeswaran et al., 2018; Tan et al., 2018; Peng et al., 2018; 2020). Despite recent success, though, model-free DRL algorithms for continuous control problems still require a large amount of training samples and computation. Due to the infinite number of feasible actions, controlling many degrees of freedom is inherently ambiguous with respect to most behaviors, resulting in challenging problems that are over specified and highly dimensional. While many methods have been proposed to help state-action exploration and/or improve sampling efficiency, including maximum entropy formulations (Haarnoja et al., 2018b;d) and using differential entropy loss terms (Mnih et al., 2016; Espeholt et al., 2018), most of such works typically relies on Gaussian-based action policies. However, the unimodal form of Gaussian distributions could still experience difficulties when facing a multi-modal reward landscape during optimization and prematurely commit to suboptimal actions (Daniel et al., 2012; Haarnoja et al., 2017).

To address such issues, people have been exploring policies more expressive than Gaussians, with a simple solution being to discretize the action space and use categorical distributions as multi-modal action policies (Andrychowicz et al., 2020; Jaśkowski et al., 2018; Tang & Agrawal, 2019). However, categorical distributions cannot be directly extended to off-policy frameworks as their sampling process is not reparameterizable. Importantly, the performance of the action space discretization depends a lot on the choice of discrete atomic actions, which are usually picked uniformly due to lack of prior knowledge. While in theory increasing the resolution of the action space can make fine control more possible, in practice this can be detrimental to the optimization problem, since the policy gradient variance increases with increasing number of atomic actions (Tang & Agrawal, 2019).

In this paper, we propose Particle Filtering Policy Network (PFPN) as a general framework to adaptively discretize high-dimensional action spaces for continuous control problems. Instead of selecting fixed samples from the continuous action space, we exploit a particle-based approach to discretize the action space dynamically during training and track a policy distribution represented as a mixture of Gaussians. The resulting policy combined with a reparameterization trick makes our approach well suited for both on-policy and off-policy policy gradient algorithms. We show that our

adaptive discretization scheme improves overall performance compared to uniform discretization and to corresponding implementations with Gaussian policies.

2 BACKGROUND

We consider a standard reinforcement learning setup where given a time horizon H and the trajectory $\tau = (\mathbf{s}_1, \mathbf{a}_1, \cdots, \mathbf{s}_H, \mathbf{a}_H)$ obtained by a transient model $\mathcal{M}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and a parameterized action policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, with $\mathbf{s}_t$ and $\mathbf{a}_t$ denoting the state and action taken at time step t, respectively, the goal is to learn the parameters θ that maximize the agent's cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r_t(\tau) \right] = \int p_\theta(\tau) r(\tau) d\tau. \tag{1}$$

Here, $p_\theta(\tau)$ denotes the state-action visitation distribution for the trajectory τ induced by the transient model $\mathcal{M}$ and the action policy π_θ, and $r(\tau) = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$ where $r(\mathbf{s}_t, \mathbf{a}_t)$ is the reward received at time step t. We can maximize $J(\theta)$ by adjusting the policy parameters θ through the gradient ascent method, where the gradient of the expected reward can be determined according to the policy gradient theorem (Sutton et al., 2000), i.e.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\cdot|\mathbf{s}_t)} \left[A_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) | \mathbf{s}_t \right]. \tag{2}$$

where A_t denotes an estimate to the reward term $r_t(\tau)$. In DRL, the estimator of A_t often relies on a separate network (critic) that is updated in tandem with the policy network (actor). This gives rise to a family of policy gradient algorithms known as actor-critic.

On-Policy and Off-Policy Actor-Critics. In on-policy learning, the update policy is also the behavior policy based on which a trajectory is obtained to estimate A_t. Common on-policy actor-critic algorithms include A3C (Mnih et al., 2016) and PPO (Schulman et al., 2017), and directly employ Equation. 2 for optimization. In off-policy learning, typically, a critic network directly learns to estimate A_t given a state-action pair (Q- or soft Q-value) such that the policy can be updated without the knowledge of a whole trajectory. This results in more sample efficient approaches as samples are reusable. Common off-policy actor-critic methods include DDPG (Lillicrap et al., 2015), SAC (Haarnoja et al., 2018b;d) and their variants (Haarnoja et al., 2017; Fujimoto et al., 2018). In DDPG, the actor network, generating a deterministic policy $\mu_\theta(\mathbf{s})$, is optimized directly by maximizing the Q-value (Silver et al., 2014), resulting in the following policy gradient update:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{B}} \left[\nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t) \nabla_\theta \mathbf{a}_t \right], \tag{3}$$

where $\mathcal{B}$ denotes a replay buffer of the collected samples, and the action $\mathbf{a}_t = \mu_\theta(\mathbf{s}_t)$ is determined by the policy network. In SAC, the stochastic policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ is optimized through entropy regularization by minimizing the divergence between the policy and the distribution of soft Q-value:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{B}} \left[\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t)) \nabla_\theta \mathbf{a}_t \right], \tag{4}$$

where $Q(\mathbf{s}_t, \mathbf{a}_t)$ denotes the soft Q-value predicted by the critic, and $\mathbf{a}_t = f_\theta(\epsilon, \mathbf{s}_t)$ is a differentiable sampling result generated by the policy network with noise ϵ. We note that both equations above require the sampled action $\mathbf{a}_t$ to be reparameterizable; otherwise, the gradient $\nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t)$ cannot be employed to update the policy network parameters θ through backpropagation.

Policy Representation. Given a multi-dimensional continuous action space, the most common choice in current DRL baselines is to model the policy π_θ as a multivariate Gaussian distribution with independent components for each action dimension (DDPG, SAC and their variants typically use Gaussian with a monotonic squashing function to stabilize the training). For simplicity, let us consider a simple case with a single action dimension and define the action policy as $\pi_\theta(\cdot|\mathbf{s}_t) := \mathcal{N}(\mu_\theta(\mathbf{s}_t), \sigma^2_\theta(\mathbf{s}_t))$. Then, we can obtain

$$\log \pi_\theta(a_t|\mathbf{s}_t) = -\frac{(a_t - \mu_\theta(\mathbf{s}_t))^2}{2\sigma^2_\theta(\mathbf{s}_t)} - \log \sigma_\theta(\mathbf{s}_t) - \log 2\pi. \tag{5}$$

Given a sampled action a_t and the estimate of cumulative rewards A_t, the optimization process based on the above expression can be imagined as that of shifting $\mu_\theta(\mathbf{s}_t)$ towards the direction of a_t if A_t is higher than the expectation, or to the opposite direction if A_t is smaller. Such an approach,

though, can easily converge to a suboptimal solution, if, for example, the reward landscape has a basis between the current location of $\mu_\theta(\mathbf{s}_t)$ and the optimal solution, or hard to be optimized if the reward landscape is symmetric around $\mu_\theta(\mathbf{s}_t)$. These issues arise due to the fact that the Gaussian distribution is inherently unimodal, while the reward landscape could be multi-modal (Haarnoja et al., 2017). Similar problems also could occur in Q-value based optimization, like DDPG, where the deterministic policy plus noise can be considered as a Gaussian with a scalar standard deviation.

3 Particle Filtering Policy Network

In this section, we describe our Particle Filtering Policy Network (PFPN) that addresses the unimodality issues from which typical Gaussian-based policy networks suffer. Our approach represents a policy as a mixture of Gaussians obtained by adaptively discretizing the action space using state-independent particles, each capturing a Gaussian distribution. The policy network, instead of directly generating actions, it is tasked with choosing particles, while the final actions are obtained by sampling from the selected particles.

3.1 Particle-based Action Policy

Let $\mathcal{P} := \{\langle p_{i,k}, w_{i,k}\rangle\}$, be a weighted set of particles where $p_{i,k} := \mathcal{N}(\mu_{i,k}, \sigma^2_{i,k})$ is the i-th particle on the k-th action dimension that represents a univariate Gaussian distribution, and $w_{i,k}$ denotes the corresponding weight of the particle. Let also $p_{i,k}(\cdot|\mu_{i,k}, \sigma_{i,k})$ denote the probability density function of the distribution $\mathcal{N}(\mu_{i,k}, \sigma^2_{i,k})$. Given $\mathcal{P}$, we define the action policy as

$$\pi^{\mathcal{P}}_\theta(\mathbf{a}_t|\mathbf{s}_t) = \prod_k \sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(a_{t,k}|\mu_{i,k}, \sigma_{i,k}), \tag{6}$$

where $a_{t,k} \in \mathbf{a}_t$ is the sampled action at the time step t for the action dimension k, and $w_{i,k}(\cdot|\theta)$ is the direct output of the policy network with parameters θ after applying a softmax operation to the output neurons for the k-th dimension satisfying $\sum_i w_{i,k}(\mathbf{s}_t|\theta) = 1$.

While the softmax operation gives us a categorical distribution defined by $w_{\cdot,k}(\mathbf{s}_t|\theta)$, the nature of the policy for each dimension is a mixture of Gaussian distributions. Given sufficient Gaussian components, the mixture of Gaussians can approximate arbitrary smooth density, though the Gaussian components are state-independent. Drawing samples from the mixture of Gaussians can be done in two steps. First, based on the weights $w_{\cdot,k}(\mathbf{s}_t|\theta)$, we perform sampling on the categorical distribution to choose a particle j_k for each dimension k, i.e.

$$j_k(\mathbf{s}_t) \sim P(\cdot|w_{\cdot,k}(\mathbf{s}_t)). \tag{7}$$

Then, we can draw actions from the Gaussian components represented by the chosen particles as:

$$a_{t,k} \sim p_{j_k(\mathbf{s}_t)}(\cdot|\mu_{j_k(\mathbf{s}_t)}, \sigma_{j_k(\mathbf{s}_t)}). \tag{8}$$

We refer to Appendix C for the action policy representation in DDPG and SAC where an invertible squashing function is applied to the Gaussian components.

3.2 Training

The proposed particle-based policy distribution is general and can be applied directly to any algorithm using the policy gradient method with Equation 2. To initialize the training, due to lack of prior knowledge, the particles can be distributed uniformly along the action dimensions with a standard deviation covering the gap between two successive particles. Without loss of generality, let us consider below only one action dimension and drop the subscript k. Then, at every training step, each particle p_i will move along its action dimension and be updated by

$$\begin{aligned} \nabla J(\mu_i, \sigma_i) &= \mathbb{E}\left[\sum_t A_t \nabla_{\substack{\mu_i \\ \sigma_i}} \log \sum_j w_j(\mathbf{s}_t|\theta) p_j(a_t|\mu_j, \sigma_j)|\mathbf{s}_t\right] \\ &= \mathbb{E}\left[\sum_t c_t w_i(s_t|\theta) \nabla_{\substack{\mu_i \\ \sigma_i}} p_i(a_t|\mu_i, \sigma_i)|\mathbf{s}_t\right], \end{aligned} \tag{9}$$

where $a_t \sim \pi^{\mathcal{P}}_\theta(\cdot|\mathbf{s}_t)$ is the action chosen during sampling, and $c_t = \frac{A_t}{\sum_j w_j(\mathbf{s}_t|\theta) p_j(a_t|\mu_j, \sigma_j)}$ is a coefficient shared by all particles on the same action dimension. Similarly, for the update of the

policy neural network, we have

$$\nabla J(\theta) = \mathbb{E}\left[\sum_t c_t p_i(a_t|\mu_i, \sigma_i)\nabla_\theta w_i(\mathbf{s}_t|\theta)|\mathbf{s}_t\right]. \tag{10}$$

From the above equations, although sampling is performed on only one particle for each given dimension, all of that dimension's particles will be updated during each training iteration to move towards or away from the location of a_t according to A_t. The amount of the update, however, is regulated by the state-dependent weight $w_i(\mathbf{s}_t|\theta)$: particles that have small probabilities to be chosen for a given state $\mathbf{s}_t$ will be considered as uninteresting and be updated with a smaller step size or not be updated at all. On the other hand, the contribution of weights is limited by the distance between a particle and the sampled action: particles too far away from the sampled action would be considered as insignificant to merit any weight gain or loss. In summary, particles can converge to different optimal locations near them during training and be distributed multimodally according to the reward landscape defined by A_t, rather than collapsing to a unimodal, Gaussian-like distribution (see Appendix E for related comparisons).

3.3 Resampling

Similar to traditional particle filtering approaches, our approach would encounter the problem of degeneracy (Kong et al., 1994). During training, a particle placed near a location at which sampling gives a low A_t value would achieve a weight decrease. Once the associated weight reaches near zero, the particle will not be updated anymore (cf. Equation 9) and become 'dead'. We adapt the idea of importance resampling from the particle filtering literature (Doucet et al., 2001) to perform resampling for dead particles and reactivate them by duplicating alive target particles.

We consider a particle dead if its maximum weight over all possible states is too small, i.e.

$$\max_{\mathbf{s}_t} w_i(\mathbf{s}_t|\theta) < \epsilon_t, \tag{11}$$

where ϵ_t is a small positive threshold number. In practice, we cannot check $w_i(s_t|\theta)$ for all possible states, but can keep tracking it during sampling and evaluate it based on the observed states collected in the last few environment steps. A target particle τ_i is drawn for each dead particle i independently from the categorical distribution based on the particle's average weight over the observed samples:

$$\tau_i \sim P\left(\cdot|\mathbb{E}_{\mathbf{s}_t}\left[w_k(\mathbf{s}_t|\theta)\right], k = 1, 2, \cdots\right). \tag{12}$$

Theorem 1. *Let $\mathcal{D}_\tau$ be a set of dead particles sharing the same target particle τ. Let also the logits for the weight of each particle k be generated by a fully-connected layer with parameters $\boldsymbol{\omega}_k$ for the weight and b_k for the bias. The policy $\pi_\theta^{\mathcal{P}}(a_t|\mathbf{s}_t)$ is guaranteed to remain unchanged after resampling via duplicating $\langle\mu_i, \sigma_i\rangle \leftarrow \langle\mu_\tau, \sigma_\tau\rangle, \forall i \in D_\tau$, if the weight and bias used to generate the unnormalized logits of the target particle are shared with those of the dead one as follows:*

$$\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_\tau; \quad b_i, b_\tau \leftarrow b_\tau - \log\left(|\mathcal{D}_\tau| + 1\right). \tag{13}$$

Proof. See Appendix B for the inference. □

Theorem 1 guarantees the correctness of our resampling process as it keep the action policy $\pi_\theta^{\mathcal{P}}(a_t|\mathbf{s}_t)$ identical before and after resampling. If, however, two particles are exactly the same after resampling, they will always be updated together at the same pace during training and lose diversity. To address this issue, we add some regularization noise to the mean value when performing resampling i.e. $\mu_i \leftarrow \mu_\tau + \varepsilon_i$, where ε_i is a small random number to prevent μ_i from being too close to its target μ_τ.

We refer to Appendix A for the pseudo code of PFPN with general policy gradient algorithms.

3.4 Reparameterization Trick

The two-step sampling method described in Section 3.1 is non reparameterizable, because of the standard way of sampling from the categorical distribution through which Gaussians are mixed. To address this issue and enable the proposed action policy applicable in Q-value based off-policy

algorithms, we consider the concrete distribution (Jang et al., 2016; Maddison et al., 2016) that generates a re-parameterized, differential continuous approximation to a categorical distribution.

Given that $\mathbf{x} \sim \text{CONCRETE}(\{w_k(\mathbf{s}_t|\theta); k = 1, 2, \cdots\}, \cdot)$ is a differentiable sampling result approximating the categorical distribution defined by $w_k(\mathbf{s}_t|\theta)$, we apply the Gumbel-softmax trick (Jang et al., 2017) to get a sampled action value as $a'(\varepsilon, \mathbf{s}_t) = \text{STOP}\left(\sum_i a_i(\varepsilon_i)\delta(i, \arg\max \mathbf{x})\right)$, where ε is a noise process, $a_i(\varepsilon_i)$ is the sample drawn from the configuration of the particle i with noise $\varepsilon_i \in \varepsilon$, $\text{STOP}(\cdot)$ is a "gradient stop" operation, and $\delta(\cdot, \cdot)$ denotes the Kronecker delta function. Then, the reparameterized sampling result can be written as follows:

$$a_\theta^{\mathcal{P}}(\varepsilon, \mathbf{s}_t) = \sum_i (a_i(\varepsilon_i) - a'(\varepsilon, \mathbf{s}_t))m_i + a'(\varepsilon, \mathbf{s}_t)\delta(i, \arg\max \mathbf{x}) \equiv a'(\varepsilon, \mathbf{s}_t), \tag{14}$$

where $m_i := x_i + \text{STOP}(\delta(i, \arg\max \mathbf{x}) - x_i) \equiv \delta(i, \arg\max \mathbf{x})$ composing a one-hot vector that approximates the sampled action from the corresponding categorical distribution.

From the above equation, the gradient can be achieved by

$$\nabla_\theta a_\theta^{\mathcal{P}} = \sum_i (a_i(\varepsilon_i) - a'(\varepsilon, \mathbf{s}_t))\nabla_\theta x_i; \nabla_{\substack{\mu_i \\ \sigma_i}} a_\theta^{\mathcal{P}} = \delta(i, \arg\max \mathbf{x})\nabla_{\substack{\mu_i \\ \sigma_i}} a_i. \tag{15}$$

4 RELATED WORK

Our approach is closely related to approaches that rely on the discretization of the action space. The discretized action space using a categorical distribution as a replacement to the original continuous space cannot be directly applied on off-policy policy gradient methods, since sampling from categorical distributions is non reparameterizable. While DQN-like approaches are very efficient for problems with discrete action spaces, such techniques cannot scale well to high-dimensional continuous action spaces due to the curse of dimensionality (Lillicrap et al., 2015). Recent work has attempted to solve this issue by using a sequential model, at the expense, though, of increasing the complexity of the state space (Metz et al., 2017). In contrast to off-policy, Q-value based approaches, action discretization with categorical distributions has been successfully applied to on-policy policy gradient methods in some high-dimensional control tasks (Andrychowicz et al., 2020; Tang & Agrawal, 2019) . All of these works discretize the action space uniformly and convert the action space to a discrete one before training. While impressive results have been obtained that allow for better performance, such a uniform discretization scheme heavily relies on the number of discretized atomic actions to find a good solution.

Our PFPN approach adopts a mixture of Gaussians as the policy to generate a discretized action scheme for continuous tasks without changing the underlying policy gradient algorithms or remodeling the problem. To that extent, our approach is complementary to recent works that focus on improving the expressiveness of the action policy through normalizing flows (Haarnoja et al., 2018a; Tang & Agrawal, 2018; Mazoure et al., 2019; Delalleau et al., 2019), where PFPN's mixture of Gaussians can be employed as a base distribution. Approaches applicable to policy gradient models that do not rely on specific properties of a policy distribution can be incorporated with our method as well, such as the use of ordinal architecture for action parameterization (Tang & Agrawal, 2019), and action space momentum as in the recent PPO-CMA (Hämäläinen et al., 2018).

PFPN uses state-independent particles that have state-dependent weights to track the policy distribution. While an early version of SAC also employed a mixture of Gaussians, such a mixture is completely state dependent, and hence it cannot support any discretization scheme. In the final version of SAC (Haarnoja et al., 2018b;d), it was replaced by a squashed Gaussian. We note that a particle-based implementation of SAC was recently proposed by Kang et al. (2019), where particles are selected based on soft Q-values without any policy network. In practice, though, such approach is only applicable to low-dimensional tasks due to the curse of dimensionality. Importantly, the particles are optimized without state-dependent importance weights, and hence quickly collapse to an average optimal location over all states. This changes the underlying maximum entropy formulation that SAC considers, and makes the algorithm lose entropy regularization.

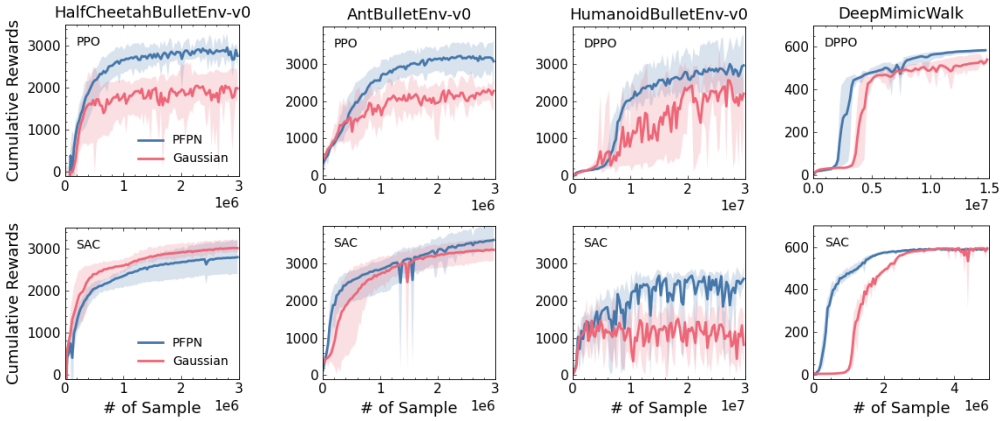


Figure 1: Performance of PPO/DPPO and SAC using our proposed PFPN (blue) and Gaussian policies (red). Solid lines report the average and shaded regions are the minimum and maximum cumulative rewards achieved in one environment episode with different random seeds during training.

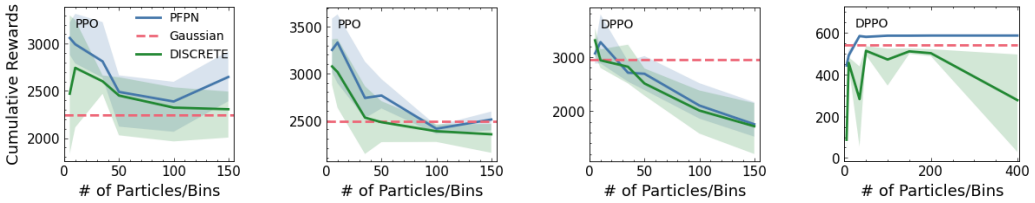
5 Experiments

The goal of our experiments is to evaluate whether existing policy gradient algorithms using our proposed PFPN can outperform the corresponding implementations with Gaussian action policies, along with comparing our adaptive action discretization scheme generated by PFPN to the fixed, uniform one. For our comparisons, we use a range of continuous torque-based control tasks from PyBullet Roboschool environments (Schulman et al., 2015). which are typically more challenging than the corresponding ones from the OpenAI Gym benchmark suite (Brockman et al., 2016). We also consider several challenging position-control tasks from the DeepMimic framework (Peng et al., 2018) where a 36-dimension humanoid agent learns a locomotion policy based on motion capture data.

5.1 Comparison to Gaussian Baselines

Figure 1 evaluates our approach on two representative policy gradient methods: PPO/DPPO (Schulman et al., 2017; Heess et al., 2017), which is a stable on-policy method that exhibits good performance, and SAC Haarnoja et al. (2018d), an off-policy method that achieves state-of-the-art performance in many tasks. The figure shows the cumulative rewards of evaluation rollouts during training using the PFPN and the Gaussian-based implementations of PPO/DPPO and SAC. We train five trials of each baseline with different random seeds that are the same across the PFPN and Gaussian versions. Evaluation was performed ten times every 1,000 training steps using deterministic action. As it can be seen in the figure, PFPN significantly improves the training performance in most of these benchmarks. Compared to the Gaussian-based implementations, our particle-based scheme achieves better or comparable final performance and usually is with faster convergence speed and small variances across multiple trials. We refer to Appendix G for additional benchmark results, as well as results obtained with A2C/A3C (Mnih et al., 2016), IMPALA (Espeholt et al., 2018), and DDPG (Lillicrap et al., 2015). In all cases, PFPN outperforms the default Gaussian-based baselines, and is more stable across different training trials. See Appendix F for all hyperparameters.

We found Gaussian policies to be very sensitive to the initial value of the mean and standard deviation in certain tasks. In DeepMimic tasks, for example, Gaussian policies depend on a carefully chosen initial standard deviation value of 0.05 per action dimension. This is a quite small value compared to the action space defined around $[-1, 1]$ per dimension, which can lead to insufficient exploration in other benchmarks. Our proposed PFPN approach has three hyperparameters, the number of particles employed, the resampling threshold ϵ_t, which determines whether a particle is dead or not, and the resampling interval, which denotes how often the resampling is performed. In practice, we found that resampling can work with a wide range of hyperparameters. As for the number of particles, Figure 2 gives an insight that PFPN can achieve its best performance using around 10 to 35 particles per action dimension, though a wider range can still work quite well outperform-



(a) HalfCheetahBulletEnv-v0 (b) AntBulletEnv-v0 (c) HumanoidBulletEnv-v0 (d) DeepMimicWalk

Figure 2: Comparison between PFPN and uniform discretization (DISCRETE) on different tasks using PPO/DPPO while varying the resolution of each action dimension. Shaded regions denote the minimum and maximum best performance over five training trials, and solid lines report the average best performance. Red dashed line denotes the average best performance of Gaussian policies over the five trials.

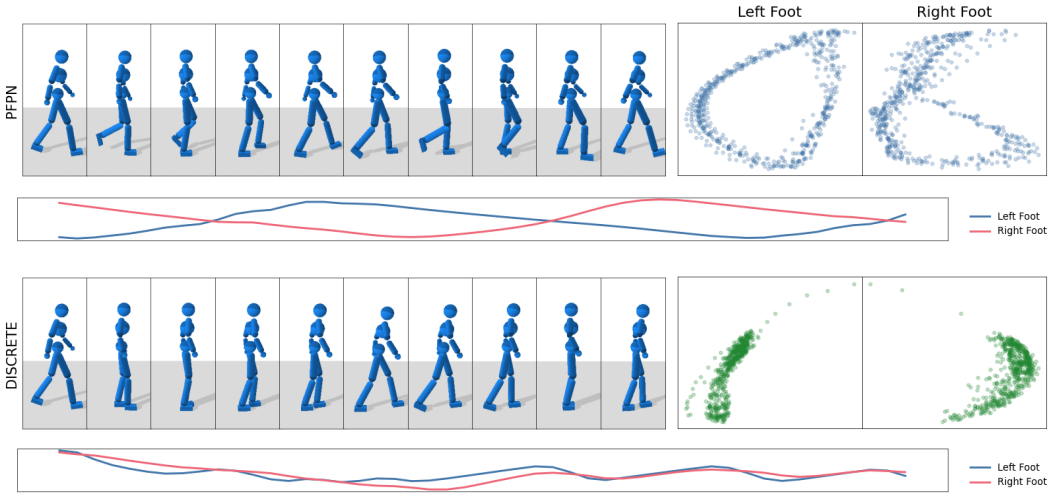


Figure 3: Comparison of motion generated by PFPN and DISCRETE in DeepMimicWalk task during one step cycle. Both PFPN and DISCRETE are trained using DPPO; PFPN uses the default hyperparameters of 35 particles per action dimension. DISCRETE employs 200 bins per action dimension. PCA embedding of the trajectories of the two feet of the agent are shown at the right, and their time expansions at the bottom of each method. While both method can train a walking agent, the DISCRETE results in an antalgic-like gait.

ing Gaussian-based policies. We refer to Appendix H for the ablation study that also compares our proposed resampling strategy to PFPN without resampling and to PFPN with random resampling.

5.2 Comparing to Fixed, Uniform Discretization.

Compared to Gaussian-based policy networks, uniformly discretizing each action dimension into a certain number of bins and sampling actions from a categorical distribution on them (DISCRETE) can work quite well in many tasks, as has been shown in recent prior work (Tang & Agrawal, 2019). However, as can be seen in Figure 2, PFPN outperforms DISCRETE across different action resolutions, i.e., number of particles in PFPN or bins in DISCRETE. Intuitively, increasing the resolution, and hence the number of atomic actions, helps both DISCRETE and PFPN. However, the more actions employed the harder the optimization problem will be for both methods (see Appendix D for theoretical analysis). This can also be verified empirically by the performance decrease in Roboschool tasks when transitioning from 10 to 150 particles. In the more complex DeepMimicWalk task, using fixed discretization with too few of atomic actions can easily skip over optimal actions. Compared to DISCRETE that needs more than 50 particles to get a relative good performance, PFPN shows its advantage by reaching better performance using only 35 particles. Additionally,

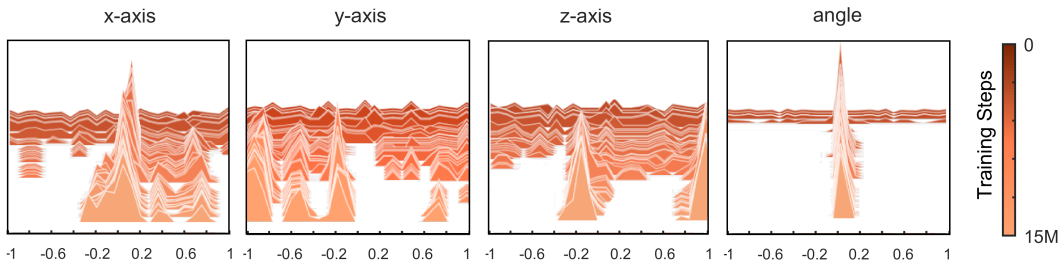


Figure 4: Evolution of how particles are distributed along four action dimensions during training of the DeepMimicWalk task with DPPO. The depicted four dimensions represent the target right hip joint position expressed in an axis-angle representation. Each action dimension is normalized between -1 and 1. Particles are initially distributed uniformly along a dimension (dark colors) and their locations adaptively change as the policy network is trained (light colors).

PFPN is more robust as it keeps stable performance as the number of particles increases beyond 35, at the cost, though of slower convergence due to the increase in policy gradient variance. We refer to Appendix H for the sensitivity analysis of PFPN with respect to the number of particles.

DeepMimic tasks use stable PD controllers (Tan et al., 2011) and exploit motion capture data to design the reward function, which is more subtle than the Roboschool reward functions that primarily measure performance based on the torque cost and agent moving speed. Given a certain clip of motion, the valid movement of a joint may be restrained in some small ranges, while the action space covers the entire movement range of that joint. Compared to uniform discretization that can place many atomic actions blindly in bad regions, PFPN optimizes the placement of atomic actions, providing a more effective discretization scheme that reaches better performance with fewer atomic actions As an example, Figure 4 shows how particles evolve during training for one of the humanoid's joints.

To highlight the control performance of the adaptive discretization scheme provided by PFPN, we compare the motion generated by PFPN and DISCRETE in Figure 3. As we can seen in the figure, PFPN generates a stable motion sequence with a nature human-like gait, while DISCRETE is stuck at a suboptimal solution and generates a walking motion sequence with an antalgic-like gait where the agent walks forward mainly through the use of its left leg. From the motion trajectories, it can also be seen that PFPN generates a more stable gait with a clear cyclic pattern. We also refer to Appendix G for more comparisons between PFPN and DISCRETE using different on-policy policy gradient methods across a range of benchmarks.

6 Conclusion

We present a general framework for learning controls in high-dimensional continuous action spaces trough particle-based adaptive discretization. Our approach uses a mixture of Gaussians represented by a set of weighted particles to track an action policy distribution during training. By introducing the reparameterization trick, the resulting particle-based policy can be adopted by both on-policy and off-policy policy gradient DRL algorithms to replace Gaussian action policies or any other action policy without changing the underlying architecture of such algorithms. Overall, our particle filtering policy network combined with existing baselines leads to better performance and sampling efficiency as compared to corresponding implementations with naive Gaussian-based policies. As a way to discretize the action space, we show that our method is more friendly to policy gradient optimization and it can significantly outperform uniform discretization schemes in complex tasks by using a smaller number of atomic actions. In our current benchmarks, we deploy particles to track each action dimension independently. This may make the problem more difficult especially for highly articulated agents. A better solution could be to account for the synergy that exists between different joints when performing a task by assigning particles to multiple joints together, which opens an exciting avenue for future work.

REFERENCES

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pp. 273–281, 2012.

Olivier Delalleau, Maxim Peter, Eloi Alonso, and Adrien Logut. Discrete and continuous action representation for practical rl in video games. *arXiv preprint arXiv:1912.11077*, 2019.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pp. 3–14. Springer, 2001.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pp. 1352–1361, 2017.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018b.

Tuomas Haarnoja, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems*, 2018c.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018d.

Perttu Hämäläinen, Amin Babadi, Xiaoxiao Ma, and Jaakko Lehtinen. Ppo-cma: Proximal policy optimization with covariance matrix adaptation. *arXiv preprint arXiv:1810.02541*, 2018.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Marco F Huber, Tim Bailey, Hugh Durrant-Whyte, and Uwe D Hanebeck. On entropy approximation for gaussian mixture random vectors. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 181–188. IEEE, 2008.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations (ICLR 2017)*, 2017.

Wojciech Jaśkowski, Odd Rune Lykkebø, Nihat Engin Toklu, Florian Trifterer, Zdeněk Buk, Jan Koutník, and Faustino Gomez. Reinforcement learning to run... fast. In *The NIPS'17 Competition: Building Intelligent Systems*, pp. 155–167. Springer, 2018.

Minjae Kang, Kyungjae Lee, and Songhwai Oh. Soft action particle deep reinforcement learning for a continuous action space. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5028–5033. IEEE, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Augustine Kong, Jun S Liu, and Wing Hung Wong. Sequential imputations and bayesian missing data problems. *Journal of the American statistical association*, 89(425):278–288, 1994.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Bogdan Mazoure, Thang Doan, Audrey Durand, R Devon Hjelm, and Joelle Pineau. Leveraging exploration in off-policy algorithms via normalizing flows. *arXiv preprint arXiv:1905.06893*, 2019.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning Agile Robotic Locomotion Skills by Imitating Animals. In *Proceedings of Robotics: Science and Systems*, 2020.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2018.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pp. 387–395, 2014.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Jie Tan, Karen Liu, and Greg Turk. Stable proportional-derivative controllers. *IEEE Computer Graphics and Applications*, 31(4):34–44, 2011.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018.

Yunhao Tang and Shipra Agrawal. Implicit policy for reinforcement learning. *arXiv preprint arXiv:1806.06798*, 2018.

Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500*, 2019.

A ALGORITHM

Algorithm 1 Policy Gradient Method using Particle Filtering Policy Network

Initialize the neural network parameter θ and learning rate α;
initialize particle parameters μ_i and σ_i to uniformly distribute particles on the action dimension;
initialize the threshold ϵ_t to detect dead particles using a small number;
initialize the value of interval n to perform resampling.
loop
  for each environment step **do**
    // Record the weight while sampling.
    $a_t \sim \pi_{\theta,\mathcal{P}}(\cdot|s_t)$
    $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{w_i(s_t|\theta)\}$
  end for
  for each training step **do**
    // Update parameters using SGD method.
    $\langle \mu_i, \sigma_i \rangle \leftarrow \langle \mu_i, \sigma_i \rangle + \alpha \nabla J(\mu_i, \sigma_i)$
    $\theta \leftarrow \theta + \alpha \nabla J(\theta)$
  end for
  for every n environment steps **do**
    // Detect dead particles and set up target ones.
    for each particle i **do**
      if $\max_{w_i \in \mathcal{W}_i} w_i < \epsilon_t$ **then**
        $\tau_i \sim P\left(\cdot|\mathbb{E}\left[w_k|w_k \in \mathcal{W}_k\right], k = 1, 2, \cdots\right)$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_i\}; \mathcal{D}_{\tau_i} \leftarrow \mathcal{D}_{\tau_i} \cup \{i\}$
      end if
    end for
    // Resampling.
    for each target particle $\tau \in \mathcal{T}$ **do**
      for each dead particle $i \in \mathcal{D}_\tau$ **do**
        // Duplicate particles.
        $\mu_i \leftarrow \mu_\tau + \varepsilon_i; \sigma_i \leftarrow \sigma_\tau$
        // Duplicate parameters of the last layer in the policy network.
        $\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_\tau; b_i \leftarrow b_\tau - \log(|\mathcal{D}_\tau| + 1)$
      end for
      $b_\tau \leftarrow b_\tau - \log(|\mathcal{D}_\tau| + 1)$
      $\mathcal{D}_\tau \leftarrow \emptyset$
    end for
    $\mathcal{T} \leftarrow \emptyset; \mathcal{W}_i \leftarrow \emptyset$
  end for
end loop

B POLICY NETWORK LOGITS CORRECTION DURING RESAMPLING

Theorem 1. *Let $\mathcal{D}_\tau$ be a set of dead particles sharing the same target particle τ. Let also the logits for the weight of each particle k be generated by a fully-connected layer with parameters $\boldsymbol{\omega}_k$ for the weight and b_k for the bias. The policy $\pi_\theta^{\mathcal{P}}(a_t|\mathbf{s}_t)$ is guaranteed to remain unchanged after resampling via duplicating $\langle \mu_i, \sigma_i \rangle \leftarrow \langle \mu_\tau, \sigma_\tau \rangle, \forall i \in D_\tau$, if the weight and bias used to generate the unnormalized logits of the target particle are shared with those of the dead one as follows:*

$$\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_\tau; \quad b_i, b_\tau \leftarrow b_\tau - \log\left(|\mathcal{D}_\tau| + 1\right). \tag{16}$$

Proof. The weight for the i-th particle is achieved by softmax operation, which is applied to the unnormalized logits L_i, i.e. the direct output of the policy network:

$$w_i(s_t) = \textsc{softmax}(L_i(s_t)) = \frac{e^{L_i(s_t)}}{\sum_k e^{L_k(s_t)}}. \tag{17}$$

Resampling via duplicating makes dead particles become identical to their target particle. Namely, particles in $\mathcal{D}_\tau \cup \{\tau\}$ will share the same weights as well as the same value of logits, say L'_τ, after

resampling. To ensure the policy identical before and after sampling, the following equation must be satisfied

$$\sum_k e^{L_k(s_t)} = \sum_{\mathcal{D}_\tau \cup \{\tau\}} e^{L'_\tau(s_t)} + \sum_{k \notin \mathcal{D}_\tau \cup \{\tau\}} e^{L_k(s_t)} \tag{18}$$

where L_k is the unnormalized logits for the k-th particle such that the weights for all particles who are not in $\mathcal{D}_\tau \cup \{\tau\}$ unchanged, while particles in $\mathcal{D}_\tau \cup \{\tau\}$ share the same weights.

A target particle will not be tagged as dead at all, i.e. $\tau \notin \mathcal{D}_k$ for any dead particle set $\mathcal{D}_k$, since a target particle is drawn according to the particles' weights and since dead particles are defined as the ones having too small or zero weight to be chosen. Hence, Equation 18 can be rewritten as

$$\sum_{i \in \mathcal{D}_\tau} e^{L_i(s_t)} + e^{L_\tau(s_t)} = (|\mathcal{D}_\tau| + 1) e^{L'_\tau(s_t)}, \tag{19}$$

Given that $e^{L_i(s_t)} \approx 0$ for any dead particle $i \in \mathcal{D}_\tau$ and that the number of particles is limited, it implies that

$$e^{L_\tau} \approx (|\mathcal{D}_\tau| + 1) e^{L'_\tau(s_t)}. \tag{20}$$

Taking the logarithm of both sides of the equation leads to that for all particles in $\mathcal{D}_\tau \cup \{\tau\}$, their new logits after resampling should satisfy

$$L'_\tau(s_t) \approx L_\tau(s_t) - \log(|\mathcal{D}_\tau| + 1). \tag{21}$$

Assuming the input of the full-connected layer who generates L_i is $\mathbf{x}(s_t)$, i.e. $L_i(s_t) = \boldsymbol{\omega}_i \mathbf{x}(s_t) + b_i$, we have

$$\boldsymbol{\omega}'_i \mathbf{x}(s_t) + b'_i = \boldsymbol{\omega}_\tau \mathbf{x}(s_t) + b_\tau - \log\left(|\mathcal{D}_\tau| + 1\right). \tag{22}$$

Then, Theorem 1 can be reached.

If we perform random sampling not based on the weights during resampling (see Appendix H), it is possible to pick a dead particle as the target particle. In that case

$$L'_\tau(s_t) \approx L_\tau(s_t) - \log(|D_\tau| + (1 - \sum_k \delta(\tau, \mathcal{D}_k))), \tag{23}$$

where $L'_\tau(s_t)$ is the new logits shared by particles in $\mathcal{D}_\tau$ and $\delta(\tau, \mathcal{D}_k)$ is the Kronecker delta function

$$\delta(\tau, \mathcal{D}_k) = \begin{cases} 1 & \text{if } \tau \in \mathcal{D}_k \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

that satisfies $\sum_k \delta(\tau, \mathcal{D}_k) \leq 1$. Then, for the particle τ, its new logits can be defined as

$$L''_\tau(s_t) \approx (1 - \sum_k \delta(\tau, \mathcal{D}_k)) L'_\tau(s_t) + \sum_k \delta(\tau, \mathcal{D}_k) L_\tau. \tag{25}$$

Consequently, the target particle τ may or may not share the same logits with those in $\mathcal{D}_\tau$, depending on if it is tagged as dead or not.

C Policy Representation with Action Bounds

In off-policy algorithms, like DDPG and SAC, an invertible squashing function, typically the hyperbolic tangent function, will be applied to enforce action bounds on samples drawn from Gaussian distributions, e.g. in SAC, the action is obtained by

$$\mathbf{a}_t(\varepsilon, \mathbf{s}_t) = \tanh(\mu_t(\mathbf{s}_t) + \text{diag}(\varepsilon)\sigma_t(\mathbf{s}_t)) \tag{26}$$

where ε is a noise vector. In DDPG, there are multiple ways to add noise but we can simply take it as the above equation but with a scalar value of σ_t.

Let $\mathbf{a}_t = \{\tanh u_{t,k}\}$ where $u_{t,k}$ is a random variable sampled to support the action on the k-th dimension. Then, the probability density function of PFPN represented by Equation 6 can be rewritten as

$$\pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) = \prod_k \sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(u_{t,k}|\mu_{i,k}, \sigma_{i,k}) / (1 - \tanh^2 u_{t,k}), \tag{27}$$

and the log-probability function becomes

$$\log \pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) = \sum_k \log \left[\sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(u_{t,k}|\mu_{i,k}, \sigma_{i,k}) - 2\left(\log 2 - u_{t,k} - \mathrm{softplus}(-2u_{t,k})\right) \right]. \tag{28}$$

D VARIANCE OF POLICY GRADIENT IN PFPN CONFIGURATION

Since each action dimension is independent to others, without loss of generality, we here consider the action a_t with only one dimension along which n particles are distributed. In order to make it easy for analysis, we set up the following assumptions: the reward estimation is constant, i.e. $A_t \equiv A$; logits to support the weights of particles are initialized equally, i.e. $w_i(s_t|\theta) \equiv \frac{1}{n}$ for all particles i and $\nabla_\theta w_1(\mathbf{s}_t|\theta) = \cdots = \nabla_\theta w_n(\mathbf{s}_t|\theta)$; particles are initialized to equally cover the whole action space, i.e. $\mu_i = \frac{i-n}{n}$, $\sigma_i^2 \approx \frac{1}{n^2}$ where $i = 1, \cdots, n$.

From Equation 10, the variance of the policy gradient under such assumptions is

$$\begin{aligned}
\mathbb{V}[\nabla_\theta J(\theta)|a_t] &= \int \frac{A_t \sum_i p_i(a_t|\mu_t,\sigma_t)\nabla_\theta w_i(\mathbf{s}_t|\theta)}{\sum_i w_i(s_t|\theta) p_i(a_t|\mu_t,\sigma_t)} a_t^2 \mathrm{d}a_t \\
&\propto \int a_t^2 \sum_i p_i(a_t|\mu_t,\sigma_t)\nabla_\theta w_i(\mathbf{s}_t|\theta)\mathrm{d}a_t \\
&\propto \sum_i \nabla_\theta w_i(\mathbf{s}_t|\theta) \int a_t^2 p_i(a_t|\mu_t,\sigma_t)\mathrm{d}a_t \\
&\overset{\propto}{\sim} \sum_i (\mu_i^2 + \sigma_i^2)\nabla_\theta w_i(\mathbf{s}_t|\theta) \\
&\propto \sum_i \frac{(i-n)^2+1}{n^2} \\
&= \frac{n}{3} + \frac{7}{6n} - \frac{1}{2} \\
&\sim 1 - \frac{3}{2n} + O(\frac{1}{n^2}).
\end{aligned} \tag{29}$$

Given $\mathbb{V}[\nabla_\theta J(\theta)|a_t] = 0$ when $n = 1$, from Equation 29, for any $n > 0$, the variance of policy gradient $\mathbb{V}[\nabla J(\theta)|a_t]$ will increase with n. Though the assumptions usually are hard to meet perfectly in practice, this still gives us an insight that employing a large number of particles may result in more challenge to optimization.

This conclusion is consistent with that in the case of uniform discretization (Tang & Agrawal, 2019) where the variance of policy gradient is shown to satisfy

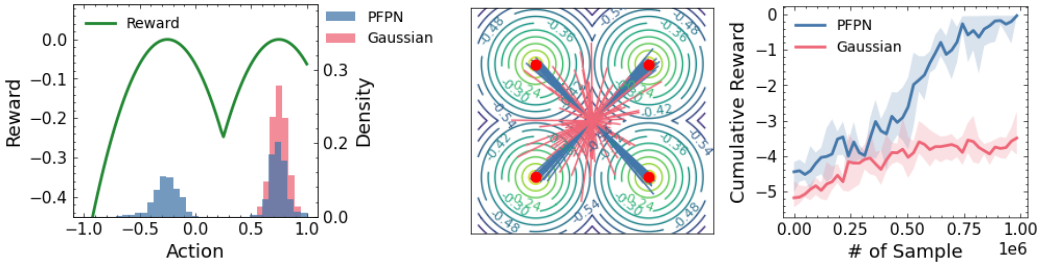
$$\mathbb{V}[\nabla_\theta J(\theta)|a_t]_{\text{DISCRETE}} \sim 1 - \frac{1}{n}. \tag{30}$$

That is to say, in either PFPN or uniform discretization scheme, we cannot simply improve the policy control capacity by employing more atomic actions, i.e. by increasing the number of particles or using more bins in the uniform discretization scheme, since the gradient variance increases as the discretization resolution increases. However, PFPN has a slower increase rate, which implies that it might support more atomic actions before performance drops due to the difficulty in optimization. Additionally, compared to the fixed, uniform discretization scheme, atomic actions represented by particles in PFPN are movable and their distribution can be optimized. This means that PFPN has the potential to provide better discretization scheme using fewer atomic actions and thus be more friendly to optimization using policy gradient.

E MULTI-MODAL POLICY

In this section, we show the multi-modal representation capacity of PFPN on special designed tasks compared to the baselines with naive Gaussian policies.

One-step Bandit. This is a simple task with one dimension action space $\mathcal{A} = [-1, 1]$. It has an asymmetric 2-peak reward landscape inversely proportional to the minimal distance to points -0.25



(a) One-step Bandit

(b) Multi-goal Navigation

Figure 5: (a) One-step bandit task with asymmetric reward landscape. The reward landscape is defined as the green line having two peaks asymmetrically at -0.25 and 0.75. The probability densities of stochastic action samples drawn from PFPN (blue) and Gaussian policy (red) are counted after training with the same number of iterations. (b) Illustration of 2D multi-goal navigation. Left: one-step trajectories generated by PFPN and Gaussian policy via stochastic sampling after training. Red dots are the four goal points placed symmetrically and the contour line depicts the reward landscape by action costs proportional to the distance to the closest goal point. The Gaussian policy is initialized around the origin. Right: The learning curve of PFPN and Gaussian policy measured by the cumulative reward across five different training seeds with PPO algorithm.

and 0.75, as the green line shown in Figure 5(a). The goal of this task is to find out the optimal points close to -0.25 and 0.75. In Figure 5(a), we show the stochastic action sample distributions of PFPN and the naive Gaussian policy after training with the same number of iterations. It is clear that PFPN captures the bi-modal distribution of the reward landscape, while the Gaussian policy gives an unimodal distribution capturing only one of reward peaks.

2D Multi-goal Navigation. In this task, the reward landscape is designed symmetrically with four goal points at $[\pm 0.5, \pm 0.5]$ and the agent has the task to reach any of the goal points. The cost (negative of the reward) function of the state $\mathcal{S} \in \mathbb{R}^2$ and action $\mathcal{A} \in \mathbb{R}^2$ is defined as the distance from the target position to the closet goal point. The naive Gaussian policy is initialized with mean value around the origin such that it is placed at the center point of the reward basin initially. Figure 5(b) shows the training performance curve and the one-step path trajectories given by PFPN and the Gaussian policy after training. While PFPN successfully learns diverse trajectories reaching all four goals, the Gaussian policy fails to learn anything due to the symmetry of the reward landscape.

As analyzed in Section 2, the optimization process of a Gaussian policy using policy gradient method is analogous to sliding the policy distribution towards samples with higher reward estimation and opposite to those with lower estimation. Hence, when facing a symmetric reward landscape, the Gaussian policy would just stay around its current location and the optimization would become hard or even impossible; and when facing an asymmetric, multimodal reward landscape, the Gaussian policy would only be able to capture only one mode at best, while PFPN can do better exploration to track multiple modes. Though in high-dimensional control problems, the reward landscape has a more complex shape, the multimodality is quite common in various, practical tasks. As a proof, Figure 4 shows the evolution of the particle distributions during training, which capture the multi-modality along different dimensions in DeepMimicWalk task. In those tasks, Gaussian policy may face the problem of premature convergence and be stuck at suboptimal solutions due to its unimodality.

F HYPERPARAMETERS

Table 1 lists the default hyperparameters used in all of our experiments. Regarding PPO and A2C, in all Roboschool tasks except for the HumanoidBulletEnv-v0 one, we use a single worker thread; for HumanoidBulletEnv-v0 and DeepMimic tasks, we exploit the advantage of distributed training

and used DPPO (synchronous PPO) and A3C (asynchronous A2C) with multiple worker threads, while IMPALA is natively multi-thread.

Certain algorithms, like A3C and IMPALA, introduce differential entropy loss to encourage exploration. However, it is not feasible to analytically evaluate the differential entropy of a mixture of Gaussians without approximation (Huber et al., 2008). As such, we use the entropy of the categorical distribution if a differential entropy term is needed during optimization.

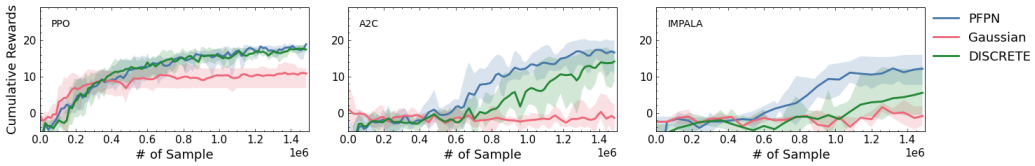
Parameter	**Value**
Shared	
optimizer	Adam Kingma & Ba (2014)
activation function	ReLU
resampling interval	25 environment episodes
dead particle detection threshold (ϵ)	0.05/# of particles per action dimension
clip range (PPO/DPPO)	0.2
GAE discount factor (PPO/DPPO, A2C/A3C, λ)	0.95
truncation level (IMPALA, $\bar{c}$, $\bar{\rho}$)	1.0
reply buffer size (SAC, DDPG)	$1 \cdot 10^6$
Roboschool Environments	
learning rate	$3 \cdot 10^{-4}$
weight initializer	Orthogonal Saxe et al. (2013)
number of neurons in hidden layers	[256, 256]
number of particle per action dimension	35 (*Humanoid*), 10 (*others*)
discount factor (γ)	0.99
mini batch size (PPO)	128
mini batch size (DPPO, *Humanoid*)	128×8 worker threads
mini batch size (A2C)	32
mini batch size (A3C, *Humanoid*)	32 (16 worker threads)
mini batch size (IMPALA)	32×2 (8 actor threads in total)
mini batch size (IMPALA, *Humanoid*)	64×4 (32 actor threads in total)
mini batch size (SAC, DDPG)	256
unroll length (PPO/DPPO)	2048
coefficient of policy entropy loss term (A2C/A3C, IMPALA)	0.01
DeepMimic Environments	
learning rate	$1 \cdot 10^{-4}$
weight initializer	Truncated Normal with std. dev. of 0.01
number of neurons in hidden layers	[1024, 512]
number of particle per action dimension	100
discount factor (γ)	0.95
mini batch size (DPPO)	32×8 worker threads
mini batch size (A3C)	32 (16 worker threads)
mini batch size (IMPALA)	64×4 (32 actor threads in total)
mini batch size (SAC)	256
unroll length (DPPO)	512
coefficient of policy entropy loss term (A3C, IMPALA)	0.00025

Table 1: Default Hyperparameters in Baseline PFPN Benchmarks

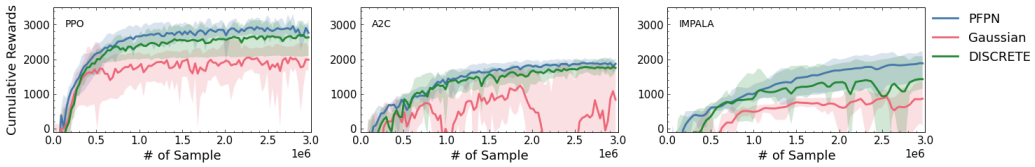
G Additional Results

G.1 Comparison to Gaussian and Fixed, Uniform Discretization Policies

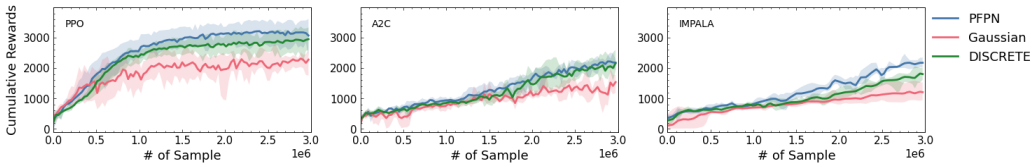
Figure 6 compares baselines that employ Gaussian policies to their PFPN counterparts on a variety of Roboschool and DeepMimic tasks and the fixed, uniform discretization scheme (DISCRETE)



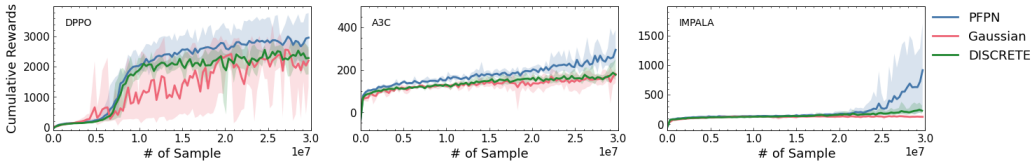
(a) ReacherBulletEnv-v0



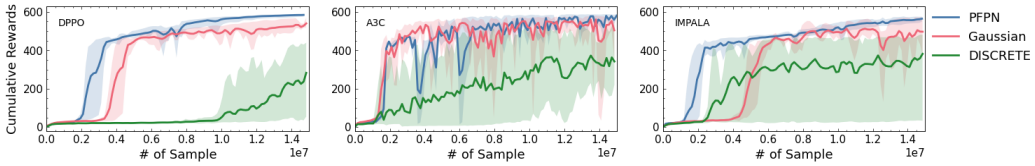
(b) HalfCheetahBulletEnv-v0



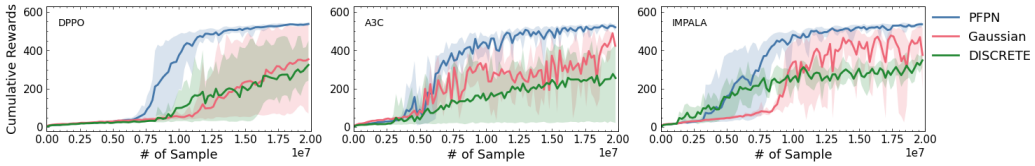
(c) AntBulletEnv-v0



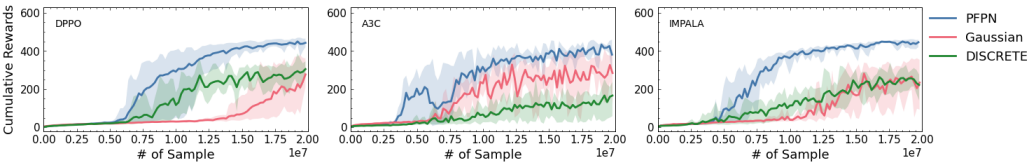
(d) HumanoidBulletEnv-v0



(e) DeepMimicWalk



(f) DeepMimicPunch



(g) DeepMimicKick

Figure 6: Training curves on continuous control tasks from the Roboschool and DeepMimic environments using on-policy policy gradient algorithms and IMPALA with v-trace correction.

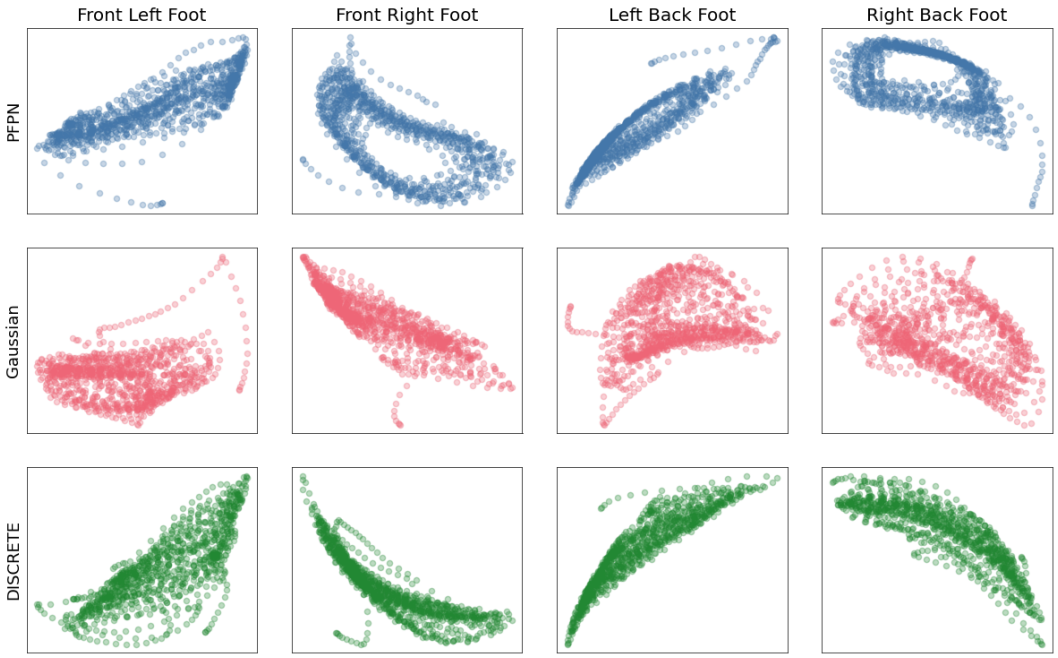


Figure 7: Motion trajectories of end effectors (four feet) of the ant agent in AntBulletEnv-v0 task with PFPN-PPO, Gaussian-based PPO and PPO with fixed, uniform discretization (DISCRETE). We apply PCA to visualize the 2D trajectories; the trajectories are measured by the relative positions of effectors with respect to the root link of the agent.

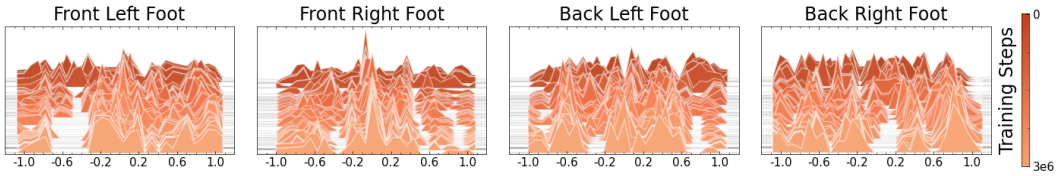
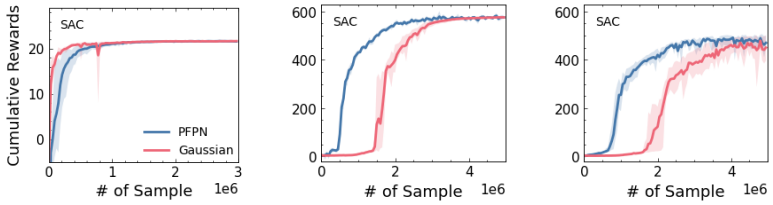


Figure 8: Evolution of particles distributed on action dimensions for the four feet joints of the agent in AntBulletEnv-v0 task during training using PPO.

with the same number of atomic actions. The results are obtained as discussed in Section 5.1, where ten evaluation trials run every 1,000 training steps using deterministic actions. PFPN outperforms Gaussian policies and fixed, uniform discretization scheme in those baselines and is more stable across different training trials.

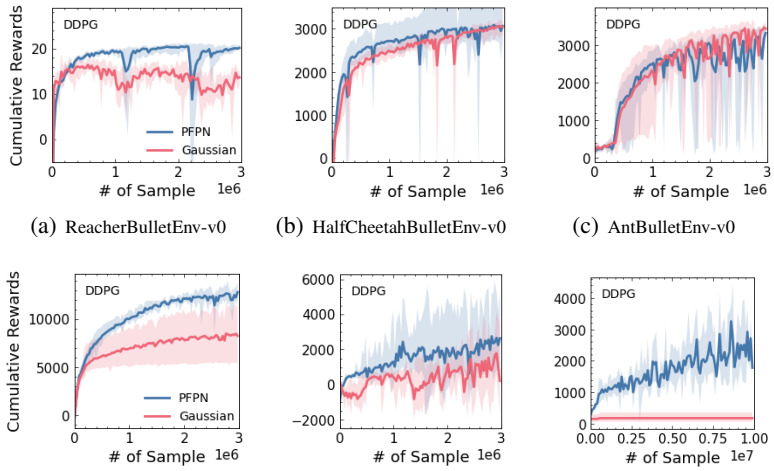
To further highlight the value that particle-based discretization adds to the action exploration problem, we also compare the motion generated by a PFPN-PPO policy in the AntBulletEnv-v0 task to the ones obtained by vanilla PPO with Gaussian action policy and that using fixed, uniform discretization scheme. We project the one-episode motion trajectories of the four end effectors (feet) of the ant agent into two dimensions in Figure 7. As it can be seen, there is a significant difference in the motion taken by the three agents, with PFPN leading to higher performance as shown in Figure 6(c). Similar to our analysis in Figure 3, Gaussian-based PPO and DISCRETE-PPO have more noise in the generated manifold, while PFPN-PPO performs robustly with a more clear cyclic motion pattern. We also show the particle evolution of the ant agent during training in Figure 8, which reflects the optimization of particle distribution.

G.2 ADDITIONAL OFF-POLICY RESULTS



(a) ReacherBulletEnv-v0 (b) DeepMimicPunch (c) DeepMimicKick

Figure 9: Learning curves on continuous control tasks using SAC in additional Roboschool and DeepMimic environments.



(a) ReacherBulletEnv-v0 (b) HalfCheetahBulletEnv-v0 (c) AntBulletEnv-v0

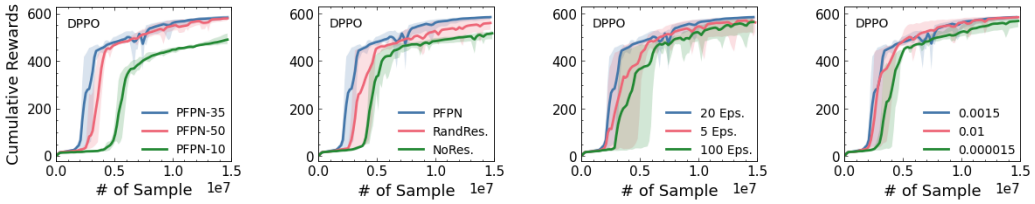
(d) HalfCheetah-v2 (e) Ant-v2 (f) Humanoid-v2

Figure 10: Learning curves on continuous control tasks using DDPG in Roboschool and OpenAI Gym environments.

We test PFPN with off-policy algorithms of SAC and DDPG, which need reparameterization tricks to run policy gradient method by state-action value. Given the state-of-the-art performance of SAC, the PFPN version of SAC performs comparably to or better than the vanilla SAC baseline and has faster convergence in most of those tasks. Considering the computation cost of running stable PD controllers (Tan et al., 2011) in DeepMimic tasks, PFPN could save hours of time during training due to its sampling efficiency. DDPG provides relatively worse performance compared to SAC and cannot work well in HumanoidBulletEnv-v0 and DeepMimic tasks. As supplement, three environments, HalfCheetah-v2, Ant-v2 and Humanoid-v2, from OpenAI Gym toolkit are introduced in Figure 10. PFPN shows its advantage in most of those tasks.

H ABLATION STUDY

Number of Particles. Intuitively, employing more particles will increase the resolution of the action space, and thus increase the control capacity and make fine control more possible. However, in Appendix D, we prove that due to the variance of policy gradient increasing as the number of particles increases, the more particles employed, the harder the optimization will be. Therefore, it may negatively influence the performance to employ too many particles. This conclusion is consistent with the benchmark results shown in Figure 2, where the DeepMimic task with more than 35 particles reaches similar best performance but at the cost of convergence speed as shown in Figure 11(a).



(a) Number of Particles (b) Resampling Strategy (c) Resampling Frequency (d) Resampling Threshold

Figure 11: Sensitivity of PFPN to the number of particles and resampling strategies and hyperparameters on DeepMimicWalk task using DPPO. (a) Comparison of PFPN using 35 particles, which is the default parameters used in above benchmarks, to that using 10 and 50 particles. (b)-(d) show the performance of PFPN with 35 particles on each action dimension but different resampling strategies or hyperparameters.

Resampling Strategies. We compare PFPN with default resampling strategy explained in Section 3.3, where the probability of drawing a target particle is proportional to its weight, to PFPN without resampling and PFPN that employs completely random resampling. It can be seen that resampling could help improve the training performance. However, the strategy of completely random resampling could lead to high variance and make the training process unstable by introducing too much uncertainty, though it still works better than the case without resampling given that particles are movable and their locations can be further optimized after resampling.

Resampling Hyperparameters. We also analyze the sensitivity of PFPN to the resampling hyperparameters: the resampling threshold ϵ_t, which determines whether a particle is dead or not, and the resampling interval, which denotes how often the resampling is performed. Results in Figure 11(c) and 11(d) show that the resampling process itself is robust and not very sensitive to these two hyperparameters. However, an extremely small value of resampling threshold or large value of resampling interval still would hurt the performance by preventing resampling and push the learning curves towards to the case of PFPN without resampling. On the other hand, a too large resampling threshold will lead alive particles to being labeled as dead, increasing the variance of the training performance. Resampling with a too small interval will incur resampling before sufficient action exploration and thus cause the same problem with a too large value of resampling interval. In our tests, we choose 20 environment episodes as the default resampling interval, and a dynamic value of resampling threshold depending on the number of particles that each action dimension has, which is around 0.0015 in DeepMimic tasks with 35 particles per action dimension.