Multi-Path Policy Optimization

Ling Pan
IIIS, Tsinghua University
Beijing, China
pl17@mails.tsinghua.edu.cn

Qingpeng Cai
Alibaba Group
Beijing, China
qingpeng.cqp@alibaba-inc.com

Longbo Huang
IIIS, Tsinghua University
Beijing, China
longbohuang@tsinghua.edu.cn

ABSTRACT

Recent years have witnessed a tremendous improvement of deep reinforcement learning. However, a challenging problem is that an agent may suffer from inefficient exploration, particularly for on-policy methods. Previous exploration methods either rely on complex structure to estimate the novelty of states, or incur sensitive hyper-parameters causing instability. We propose an efficient exploration method, Multi-Path Policy Optimization (MPPO), which does not incur high computation cost and ensures stability. MPPO maintains an efficient mechanism that effectively utilizes a population of diverse policies to enable better exploration, especially in sparse environments. We also give a theoretical guarantee of the stable performance. We build our scheme upon two widely-adopted on-policy methods, the Trust-Region Policy Optimization algorithm and Proximal Policy Optimization algorithm. We conduct extensive experiments on several MuJoCo tasks and their sparsified variants to fairly evaluate the proposed method. Results show that MPPO significantly outperforms state-of-the-art exploration methods in terms of both sample efficiency and final performance.

KEYWORDS

Deep reinforcement learning; Policy optimization

1 INTRODUCTION

In reinforcement learning, an agent seeks to find an optimal policy that maximizes long-term rewards by interacting with an unknown environment. Policy-based methods, e.g., [9, 24, 26], optimize parameterized policies by gradient ascent on the performance objective. Directly optimizing the policy by vanilla policy gradient methods may incur large policy changes, which can result in performance collapse due to unlimited updates. To resolve this issue, Trust Region Policy Optimization (TRPO) [34] and Proximal Policy Optimization (PPO) [36] optimize a surrogate function in a conservative way, both being on-policy methods that perform policy updates based on samples collected by the current policy. These on-policy methods have the desired feature that they generally achieve stable performance. In contrast, off-policy learning, where policies are updated according to samples drawn from a different policy, e.g., using an experience replay buffer [15], can often suffer

from practical convergence and stability issue [12, 13]. However, as on-policy methods learn from what they collect, they can particularly suffer from insufficient exploration ability, especially in sparse environments [5]. Thus, although TRPO and PPO start from a stochastic policy, the randomness in the policy decreases quickly during training. As a result, they can converge too prematurely to bad local optima in high-dimensional or sparse reward tasks.

Indeed, how to achieve efficient exploration is challenging in deep reinforcement learning. There has been recent progress in improving exploration ranging from count-based exploration [8, 29, 40], intrinsic motivation [1, 18, 30], to noisy networks [7, 31]. However, these methods either introduce sensitive parameters that require careful tuning on tasks [22], or require learning additional complex structures to estimate the novelty of states. Another line of research [17, 25] proposes to encourage exploration by augmenting the objective function with a diversity term that measures the distance of current and prior policies. Yet, the distance between the current policy and past polices can be small for trust-region methods where the policy update is controlled, and limits the applicability of the diversity-driven approaches.

Vanilla evolutionary algorithms [38], e.g., population-based methods [42], on the other hand, exhibit great exploration ability and stability with the maintenance of a population of agents, which are able to collect diverse samples [3, 22]. However, evolutionary algorithms are sample-inefficient compared with deep reinforcement learning approaches [5], as they learn from the whole episode instead of single steps [37], and does not exploit the powerful gradient information [22].

Recent research has shown great potential in combining deep reinforcement learning with population-based methods to reap the best from both families of algorithms [19, 21–23, 32], which mainly focuses on off-policy learning. These approaches generally maintains a population of agents, and each of them interacts with the environment to collect experiences. The key to success is that these diverse experiences are stored in a shared experience replay buffer, which can be utilized by off-policy algorithms. However, it is sample-inefficient to apply population-based methods to on-policy learning. This is due to the nature of on-policy methods, where the policy can only be updated by samples collected by itself. Therefore, rolling out all policies in the population at each iteration as in [21, 22, 32] can be inefficient as each policy cannot exploit other policies' experiences. In addition, each interaction with the environment can be expensive [2].

To enable an effective and efficient combination of on-policy reinforcement learning algorithms and population-based methods, in this paper, we propose a novel method, Multi-Path Policy Optimization (MPPO), which improves exploration for on-policy algorithms using multiple paths. Here, a path refers to a sequence of policies generated during the course of policy optimization starting from
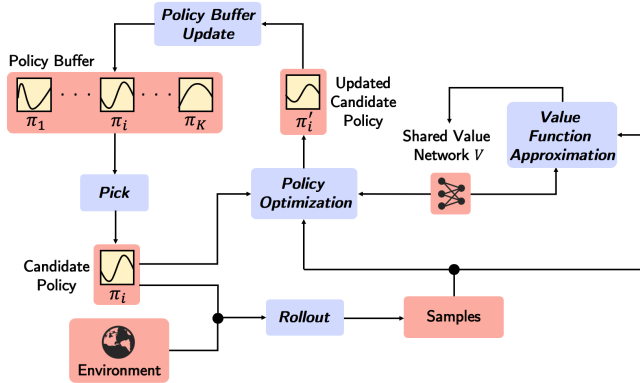
Figure 1: High-level schematic of MPPO.

a single policy. Figure 1 demonstrates the high-level schematic of MPPO, which has four main components, i.e., *pick and rollout*, *value function approximation*, *policy optimization*, and *policy buffer update*. Specifically, MPPO starts with K different policies randomly initialized in the policy buffer, and a shared value network. At each iteration, a candidate policy is picked from the policy buffer according to a picking rule, defined as a weighted combination of performance and entropy, introduced to enable a trade-off between the exploration and exploitation. Then, the picked policy interacts with the environment by rollouts to collect samples. The shared value network is updated based on these samples to approximate the value function. The picked policy is updated by policy optimization according to the samples and the shared value network. Finally, the improved picked policy updates the policy buffer by replacing itself, in order to retain the diversity of the policy buffer.

With this scheme, MPPO maintains K policy paths, which increases the exploration ability during training. Different policy paths provide diverse experiences for the shared value network to enable a better estimation [27], which yields a better signal for telling how well each state is. With a better estimated value function, the picked updated by policy optimization are more able to collect trajectories with higher rewards. Therefore, MPPO can provide better guidance for the picked policy, which MPPO aims to optimize, to explore states and actions that were not known to have high rewards previously. Moreover, since only one candidate policy is picked and optimized at each iteration, our method does not incur much computational cost compared with the base policy optimization method.

A critical component of MPPO is the picking rule, which favors to select the policy that is most desirable to rollout and to optimize at each iteration, i.e., the one with good performance while being explorative simultaneously. We prove that when MPPO switches to an explorative policy, the performance variation of picked polices can be bounded and controlled. This is a useful feature that ensures smooth policy transition. We also empirically validate that the potential variation is small, and the picked policy converges to one single policy, which ensures the stability.

We apply MPPO to two widely adopted on-policy algorithms, TRPO [34] and PPO [36], and conduct extensive experiments on several continuous control tasks based MuJoCo [41]. Experimental

results demonstrate that our proposed algorithms, MP-TRPO and MP-PPO, provide significant improvements over state-of-the-art exploration methods, in terms of sample efficiency and final performance without incurring high computational cost. We also analyze the effect of each component in our methodology and investigate the critical advantages of the proposed picking rule and policy buffer update strategy.

The main contributions can be summarized as follows:

- We propose a novel methodology utilizing a population of policies to tackle the exploration bottleneck of on-policy reinforcement learning algorithms, which is efficient and effective without introducing much computation costs.
- We give a theoretical guarantee of stable performance of Multi-Path TRPO (MP-TRPO).
- MPPO can be readily applied given any baseline on-policy algorithm. We validate MPPO to two popular on-policy algorithms, TRPO and PPO, and conduct extensive evaluation on a wide range of MuJoCo tasks. Results show that MPPO outperforms state-of-the-art exploration methods.

2 PRELIMINARIES

A Markov decision process (MDP) is defined by $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}, \mathcal{A}$ denote the set of states and actions, $p(s'|s, a)$ the transition probability from state s to state s' under action a, $r(s, a)$ the corresponding immediate reward, and $\gamma \in [0, 1)$ the discount factor. The agent interacts with the environment by its parameterized policy π_θ, with the goal to learn the optimal policy that maximizes the expected discounted return $J(\pi_\theta) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \pi_\theta]$.

Trust Region Policy Optimization (TRPO) [34] learns the policy parameter by optimizing a surrogate function in a conservative way. Specifically, it limits the stepsize towards updating the policy using a trust-region constraint, i.e.,

$$\max_{\theta} \quad \mathcal{L}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right] \quad (1)$$

$$\text{s.t.} \quad \mathbb{E}_t \left[D_{KL} \left(\pi_\theta(\cdot|s_t) || \pi_{\theta_{\text{old}}}(\cdot|s_t) \right) \right] \leq \delta, \quad (2)$$

where $\mathbb{E}_t[\dots]$ is the empirical average over a finite batch of samples, $A_t^{\pi_{\theta_{\text{old}}}}(s_t, a_t) = Q_t^{\pi_{\theta_{\text{old}}}}(s_t, a_t) - V_t^{\pi_{\theta_{\text{old}}}}(s_t)$, and $Q_t^{\pi_{\theta_{\text{old}}}}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}} \left[V_t^{\pi_{\theta_{\text{old}}}}(s_{t+1}) \right]$. One desired feature of TRPO is that it guarantees a monotonic policy improvement, i.e., the policy update step leads to a better-performing policy during training. However, it is not computationally efficient as it involves solving a second-order optimization problem using conjugate gradient.

Proximal Policy Optimization (PPO) [36] is a simpler method only involving first-order optimization using stochastic gradient descent. PPO maximizes a KL-penalized or clipped version of the objective function to ensure stable policy updates, where the clipped version is more common and is reported to perform better than the KL-penalized version. Specifically, the objective for the clipped version is to maximize

$$\mathcal{L}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) = \mathbb{E}_t \left[\min(r_t(\pi_{\theta_{\text{old}}}, \pi_\theta) A_t, \text{clip}(r_t(\pi_{\theta_{\text{old}}}, \pi_\theta), 1 - \epsilon, 1 + \epsilon) A_t) \right], \quad (3)$$

where $r_t(\pi_{\theta_{\text{old}}}, \pi_\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denotes the probability ratio, and ϵ is the parameter for clipping.

3 MULTI-PATH POLICY OPTIMIZATION

3.1 A Motivating Example

Figure 2(a) shows a challenging environment Maze of size 21×21 with sparse rewards, where black lines in the middle represent walls. The agent always starts at S located on the lower left corner of the maze with the goal of reaching the destination G in the lower right corner, where the maximum length of an episode is 1000. A reward of +1 is given only when the agent reaches G, and 0 otherwise. The experimental setting in Maze is the same as in Section 4.1.
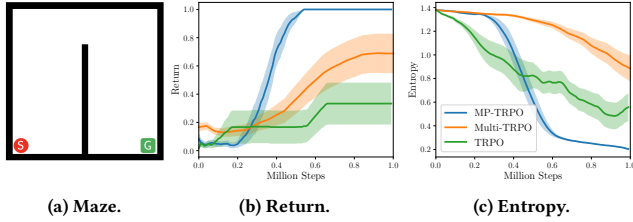

| (a) Maze. | (b) Return. | (c) Entropy. |

Figure 2: Performance and entropy comparisons on Maze.

We compare three schemes in this environment, i.e., TRPO, MP-TRPO (MPPO applied to TRPO) and Multi-TRPO (training a population of policies and picking the best one). For fair comparison, all methods use the same amount of samples during training. As shown in Figure 2(b), the agent can fail to reach the goal state under TRPO. Figure 3(a) shows the resulting state visitation density under TRPO after training for 1 million steps. Specifically, the brightness of a region in Figure 3 indicates the number of times the agent visits that region, i.e., the brighter the region is, the more times the agent visits that region. It can be seen that the agent can only explore a very limited area in the maze and mainly stays in the left side. It is also worth noting that simply training the ensemble of policies and choosing the best, i.e., Multi-TRPO, also fails to consistently find the destination. Although it is able to search a larger region, it still mostly re-explores the left part as shown in Figure 3(b). In contrast, MP-TRPO can always successfully reach the destination after 0.6 million steps while others fail. As illustrated in Figure 3(c), it is capable to bypass the wall and explore both sides of the maze.
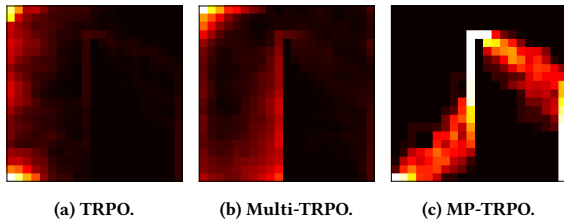

| (a) TRPO. | (b) Multi-TRPO. | (c) MP-TRPO. |

Figure 3: State visitation comparisons.

This is because TRPO suffers from insufficient exploration, and the entropy of the policy trained with TRPO decreases quickly as the policy is being optimized, as shown in Figure 2(c). Multi-TRPO maintains greater exploration ability with the population of policies.

However, recall that all three schemes consume the same amount of samples for training. As it rolls out all policies at each iteration, the performance improvement of any single policy in the population is limited compared with MP-TRPO. Indeed, for Multi-TRPO, the acquirement of diverse samples from the policy buffer comes at the expense of insufficient training of each policy under limited number of samples. This is because on-policy algorithms cannot utilize experiences from other policies, and can only update the policy based on samples collected by itself. On the other hand, during the training process, MP-TRPO optimizes the policy while simultaneously maintaining enough exploration ability.

We next systematically describe our proposed method and the motivation behind it, to illustrate why MPPO helps to improve exploration.

3.2 Method

The main idea of MPPO is summarized as follows. The policy buffer is initialized with K random policies, and a shared value network V is also randomly initialized. At each iteration t, a candidate policy π_{it} is picked from the policy buffer $\pi_t = (\pi_{1t}, ..., \pi_{Kt})$, which is used as the rollout policy to interact with the environment to generate a set of samples. The collected samples contribute to updating the shared value network for value function approximation. The candidate policy is optimized according to the collected samples based on the shared value network. Finally, the improved candidate policy π'_{it} updates the policy by replacing itself.

Specifically, the key components of the Multi-Path Policy Optimization method are as follows:

3.2.1 Pick and rollout. From previous analysis of Multi-TRPO, although a population of policies can bring diverse samples, policies in the population cannot directly exploit others' experiences in on-policy learning. Therefore, it is unnecessary to rollout all policies in the population to interact with the environment for samples collection at each iteration. In order to guarantee sample efficiency, we propose to pick a candidate policy from the current policy buffer at each iteration.

One common way is to pick a candidate policy randomly as in [28]. However, this picking rule fails to fully utilize the policy buffer. This is because it can hardly provide guidance for the agent to pick the policy that is most desirable to rollout and to optimize, as each interaction with the environment can be expensive [2].

The picking rule for MPPO is to choose the policy π_{it} with highest score f_i, which takes into account both performance and entropy as defined in Eq. (4), i.e.,

$$\forall k, \ f_k(\pi_t) = (1 - \alpha)\hat{J}_k(\pi_t) + \alpha\hat{\mathcal{H}}_k(\pi_t), \tag{4}$$

where $\hat{J}$ and $\hat{\mathcal{H}}$ denote the normalized performance and entropy according to min-max normalization as in Eq. (5).

$$\forall k, \ \hat{J}_k(\pi_t) = \frac{J_k(\pi_t) - \min_m J_m(\pi_t)}{\max_m J_m(\pi_t) - \min_m J_m(\pi_t)},$$
$$\hat{\mathcal{H}}_k(\pi_t) = \frac{\mathcal{H}_k(\pi_t) - \min_m \mathcal{H}_m(\pi_t)}{\max_m \mathcal{H}_m(\pi_t) - \min_m \mathcal{H}_m(\pi_t)}. \tag{5}$$

In Eq. (5), $J_k(\pi_t)$, $\mathcal{H}_k(\pi_t)$ denote the performance and entropy of policy π_{kt} respectively, where we use Shannon entropy defined

---

Algorithm 1: Multi-Path Policy Optimization algorithm.

Input: Initial policy buffer $\pi_0 = (\pi_{10}, ..., \pi_{K0})$ for K initial policies with parameters $\theta_0 = (\theta_{10}, ..., \theta_{K0})$, initial value function V_{ϕ_0}

1 **for** $t = 0, 1, ...$ **do**

2   Normalize $J(\pi_t)$ and $\mathcal{H}(\pi_t)$ according to Eq. (5) by $\forall k$, $\hat{J}_k(\pi_t) = \frac{J_k(\pi_t) - \min_m J_m(\pi_t)}{\max_m J_m(\pi_t) - \min_m J_m(\pi_t)}$, $\hat{\mathcal{H}}_k(\pi_t) = \frac{\mathcal{H}_k(\pi_t) - \min_m \mathcal{H}_m(\pi_t)}{\max_m \mathcal{H}_m(\pi_t) - \min_m \mathcal{H}_m(\pi_t)}$.

3   Compute scores $f_k(\pi_t)$ according to Eq. (4) by $\forall k$, $f_k(\pi_t) = (1 - \alpha)\hat{J}_k(\pi_t) + \alpha\hat{\mathcal{H}}_k(\pi_t)$

4   Select the candidate policy π_{it} where $i = \arg\max_k f_k(\pi_t)$

5   Collect set of trajectories $\mathcal{D}$ by rolling out policy π_{it} in the environment

6   Evaluate the performance of the candidate policy $J_i(\pi_t)$ based on the collected trajectories $\mathcal{D}$

7   Update the value function V_{ϕ_t} by regression on mean-squared error

8   Compute advantage estimates $A_{\pi_{it}}$ using generalized advantage approach [35] based on V_{ϕ_t}

9   Update the candidate policy parameter from $\pi_{\theta_{it}}$ to $\pi_{\theta_{i(t+1)}}$ by the base policy optimization method, e.g., TRPO or PPO

10   Compute the performance gain $\mathcal{G}_{\pi_{\theta_{it}}}(\pi_{\theta_{i(t+1)}})$ according to Eq. (1) by $\hat{\mathbb{E}}_t \left[\frac{\pi_{\theta_{i(t+1)}}(a_t|s_t)}{\pi_{\theta_{it}}(a_t|s_t)} A_t^{\pi_{\theta_{it}}}(s_t, a_t) \right]$

11   Update the policy buffer by $\pi_{t+1} = (\pi_{1t}, ..., \pi_{it}', ..., \pi_{Kt})$ and the entropy buffer accordingly

12   Update the performance buffer by $J(\pi_{t+1}) = \left(J_1(\pi_t), ..., J_i(\pi_t) + \mathcal{G}_{\pi_{\theta_{it}}}(\pi_{\theta_{i(t+1)}}), ..., J_K(\pi_t) \right)$

---

by $\mathcal{H}_k(\pi_t) = \mathbb{E}_{\pi_{kt}}[-\log \pi_{kt}(a|s)]$, and other forms of entropy can also be used, e.g., Tsallis entropy [4].

The picking rule favors to pick the policy that is most desirable to rollout and to optimize, i.e., the one with good performance while being explorative simultaneously, which is a critical component of MPPO. In Eq. (4), α provides the trade-off between exploration and exploitation. Note that a criterion focusing only on the performance cannot make good use of the policy buffer, as it tends to pick the policy updated in last iteration. Therefore, it leads to a similar optimization process as that of single-path, which also suffers from insufficient exploration. Considering the entropy term encourages exploring new behaviors. However, if one always pick the policy with the maximum entropy, it fails to exploit learned good behaviors. Our weighted rule is designed to strike for a good tradeoff between exploration and exploitation.

3.2.2 Value function approximation. Samples collected by the candidate policy contribute to updating the shared value network to approximate the value function by minimizing the mean-squared error: $\frac{1}{N} \sum_{n=1}^{N} (r_n + \gamma V_{\phi_t}(s_{n+1}) - V_{\phi_t}(s_n))^2$. During the course of training with MPPO, the shared value network exploits diverse samples collected by policies that are most desirable to be picked from the diverse policy buffer at each iteration. In this way, it can better estimate the value function compared with that of single-path. Therefore, it provides more information for the advantage function to distinguish good or bad actions, which is critical and helpful for policy optimization.

3.2.3 Policy optimization. At each iteration t, only the candidate policy π_{it} is optimized using a base policy optimization method, according to samples collected by itself and the shared value network. [1] The objective of policy optimization is to maximize the expected advantages over the policy distribution, where the estimated policy gradient is $\frac{1}{N} \sum_{n=1}^{N} \nabla_{\theta_{it}} \log \pi_{\theta_{it}}(a_n|s_n)A_{\pi_{it}}(s_n, a_n)$, given a batch of samples $\{(s_n, a_n, r_n, s_{n+1})\}$. As discussed in the previous section, MPPO enables a better estimation of the advantage function with

[1]Note that MPPO aims to optimize the picked policy instead of all policies in the population.

its mechanism utilizing the policy buffer. Therefore, policy optimization drives each picked policy to explore previously unseen good states and actions.

3.2.4 Policy buffer update. Given the optimized policy at current iteration, the policy buffer needs to be updated. A common way to update the policy buffer is to replace the worst policy in the policy buffer with the improved policy, as usually used in evolutionary-based methods for off-policy learning [22]. However, this updating scheme quickly loses the diversity of the policy buffer, and leads to a set of very similar policies ultimately, which results in a low exploration level, and will be further validated in Section 4.4. In MPPO, the updated policy will be added to the policy buffer by replacing the candidate policy itself, i.e., $\pi_{t+1} = (\pi_{1t}, ..., \pi_{it}', ..., \pi_{Kt})$, which is able to maintain the diversity of the policy buffer.

The overall algorithm for the MPPO method is shown in Algorithm 1. It is crucial to note that only the candidate policy interacts with the environment for sample collection, based on which both the candidate policy and the shared value network are updated.

3.3 Multi-Path Trust Region Policy Optimization

We first apply our proposed MPPO method to a widely adopted on-policy algorithm TRPO [34], and obtain the resulting Multi-Path Trust Region Policy Optimization (MP-TRPO) algorithm.

Specifically, the update for the candidate policy is by backtracking line search with

$$\theta_{i(t+1)} = \theta_{it} + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_{it}^T \hat{H}_{it} \hat{x}_{it}}} \hat{x}_{it}, \tag{6}$$

where $\hat{x}_{it} = \hat{H}_{it}^{-1} \hat{g}_{it}$ is computed by the conjugate gradient algorithm, and $\hat{g}_{it} = \nabla_\theta \mathcal{L}_{\pi_{\theta_{it}}}(\pi_\theta)|_{\theta=\theta_{it}}$ is the estimated policy gradient. Note that the performance of the updated policy $J_i(\pi_{t+1})$ is estimated by $J_i(\pi_t) + \mathcal{G}_{\pi_{\theta_{it}}}(\pi_{\theta_{i(t+1)}})$. Therefore, MP-TRPO does not require extra samples to evaluate the updated policy.

During the course of policy optimization, if the same policy is picked as in last iteration, MP-TRPO guarantees a monotonic improvement of the policy picked in current iteration over that in last iteration by [34]. On the other hand, if a policy that is more explorative but the performance is not as good as that in last iteration is picked, it may lead to a temporary performance drop. In Theorem 3.1, we show that such a performance drop can be bounded, ensuring a smooth policy transition.

THEOREM 3.1. *Let i, j denote the indexes of policies that are picked at timestep $t, t+1$, respectively. Denote the improvement of $J_i(\pi_{t+1})$ over $J_i(\pi_t)$ as σ_t. Then, the following bound holds for $0 \le \alpha < 1$: $J_j(\pi_{t+1}) - J_i(\pi_t) \ge \frac{-\alpha}{1-\alpha} [\max_k J_k(\pi_{t+1}) - \min_k J_k(\pi_{t+1})] + \sigma_t$.*

PROOF. As π_j is the policy selected at timestep $t + 1$, we have

$$f_j(\pi_{t+1}) > f_i(\pi_{t+1}). \qquad (7)$$

Thus,

$$\hat{J}_j(\pi_{t+1}) - \hat{J}_i(\pi_{t+1}) > \frac{-\alpha}{1-\alpha}\left(\hat{\mathcal{H}}_j(\pi_{t+1}) - \hat{\mathcal{H}}_i(\pi_{t+1})\right) \ge \frac{-\alpha}{1-\alpha}. \qquad (8)$$

According to the min-max normalization, we have

$$\hat{J}_j(\pi_{t+1}) = \frac{J_j(\pi_{t+1}) - \min_k J_k(\pi_{t+1})}{\max_k J_k(\pi_{t+1}) - \min_k J_k(\pi_{t+1})}. \qquad (9)$$

Then, we obtain

$$J_j(\pi_{t+1}) - J_i(\pi_{t+1}) \ge \frac{-\alpha}{1-\alpha}\left[\max_k J_k(\pi_{t+1}) - \min_k J_k(\pi_{t+1})\right]. \qquad (10)$$

According to the monotonic improvement theorem [34], we have

$$J_j(\pi_{t+1}) - J_i(\pi_t) \ge \frac{-\alpha}{1-\alpha}\left[\max_k J_k(\pi_{t+1}) - \min_k J_k(\pi_{t+1})\right] + \sigma_t. \qquad (11)$$

$\square$

Theorem 3.1 shows that although there may be a temporary performance drop due to switching to a more explorative policy, such a sacrifice is bounded by an α-related term and the difference of the performance of the best and the worst policies in current policy buffer.

Figure 4(a) shows the learning process of MP-TRPO on Maze (Figure 2(a)) for a single seed. In the beginning, MP-TRPO may pick different policies to collect samples and to optimize according to the picking rule, which provides diverse samples for updating the shared value network to better estimate the value function. This phase corresponds to the shaded region in Figure 4(a), which leads to the fact that the performance gap between the best and the worst policies in the policy buffer is small (Figure 4(b)). Note that we use a fixed value of α to be 0.1 in our experiments. Therefore, the temporary performance drop is very small by Theorem 3.1. Note that a better estimation of the value function drives the optimization of the picked policy. In the end, MP-TRPO will converge to picking a single policy, in which case the performance of the picked policy will be monotone increasing.

We remark here that our method maintains good performance throughout the policy optimization process, while bringing the advantage of better exploration.
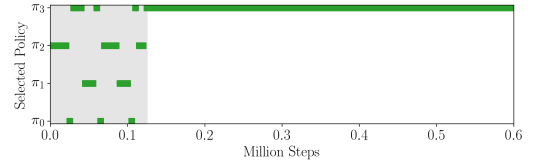
(a) Visualization of the course of policy picking.
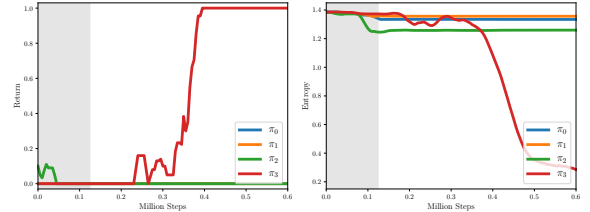
(b) Return of each policy.   (c) Entropy of each policy.

Figure 4: Learning details of MP-TRPO on Maze.

3.4 Multi-Path Proximal Policy Optimization

We also apply MPPO to another on-policy algorithm PPO [36], and obtain the MP-PPO algorithm. To be specific, the candidate policy's parameter is updated by stochastic gradient descent according to

$$\theta_{i(t+1)} = \theta_{it} + \eta\hat{g}_{it}, \qquad (12)$$

where η is the learning rate, and $\hat{g}_{it} = \nabla_\theta \mathcal{L}_{\pi_{\theta_{it}}}(\pi_\theta)|_{\theta=\theta_{it}}$ is the policy gradient estimated according to Eq. (3).

4 EXPERIMENTS

We conduct extensive experiments to investigate the following key questions:

- How does MPPO compare with single-path policy optimization and state-of-the-art exploration methods?
- What is the effect of the number of paths K and the weight α?
- Which component of MPPO is critical for the improvement of the exploration ability?
- Is MPPO generally applicable given a baseline on-policy reinforcement learning algorithm to encourage exploration?

4.1 Experimental Setup

We evaluate MPPO on several continuous control environments simulated by the MuJoCo framework [41], which is a standard and widely-used benchmark for evaluating deep reinforcement learning algorithms [6]. MuJoCo tasks exhibit dense rewards, where the agent receives a reward at each step. To better examine the exploration ability of our method, we further conduct evaluation in some more challenging variants of the original environments with sparse rewards [8, 10, 18, 20, 31]. For example, in SPARSEDOUBLEPENDULUM, a reward of +1 is given only when the agent reaches the goal that it swings the double pendulum upright, and 0 otherwise. Each algorithm is run with 6 different random seeds (0-5), and the performance is evaluated for 10 episodes every 10, 000 steps. Note that the performance of MPPO is evaluated by the picked policy. The averaged return in evaluation is reported as the solid line,

with the shaded region denoting a 75% confidence interval. For fair comparisons, the hyper-parameters for all comparing algorithms are set to be the same as the best set of hyper-parameters reported in [16].

4.2 Baselines

To comprehensively study the MP-TRPO algorithm, we compare it with six baselines. For fair comparison, all methods use the same amount of N samples during the course of policy optimization.

- **TRPO [34].** Vanilla single-path TRPO algorithm.
- **Curiosity-TRPO [30].** The curiosity-driven approach, which is a state-of-the-art method for exploration by augmenting the reward function with learned intrinsic rewards.
- **Diversity (Div)-TRPO [17].** The diversity-driven approach, which is also a state-of-the-art exploration method that augments the loss function of the policy with the distance of current policies and prior policies.
- **Multi-TRPO.** A baseline method that trains multiple (K) single-path TRPO with a shared value network and chooses the best one. We compare with the method to isolate the effect of the policy ensemble.
- **Multi-TRPO (Independent).** A baseline method training K single-path TRPO, where each policy has its own value network, resulting in K independent value networks in total. We compare with the method is to validate the effect of the shared value network.
- **MP-TRPO (ReplaceWorst).** A variant of MP-TRPO which updates the policy buffer by replacing the worst policy with the improved candidate policy. We compare with the method to evaluate the importance of the replacement strategy for updating the policy buffer.

We also verify the effectiveness of the picking rule by using different weights of α in MP-TRPO.

Then, we apply our proposed multi-path policy optimization mechanism to another baseline policy optimization method, PPO, to demonstrate the general applicability of the MPPO method, and conduct similar evaluation.

4.3 Ablation Study

4.3.1 The effect of the number of paths K. Figure 5 shows the performance of MP-TRPO and MP-PPO with varying K on SPARSE-DOUBLEPENDULUM. The K value trades off the diversity of the policy buffer and sample efficiency. A larger K maintains a greater diversity, but may require more samples to learn as there are more policies to be picked and to be optimized in early periods of learning. Indeed, there is an intermediate value for K that provides the best trade-off. We find that MP-TRPO with $K = 8$ achieves the best performance and thus we fix K to be 8 on all environments. For MP-PPO, a relatively smaller $K = 2$ is sufficient and performs best. This is because PPO itself exhibits greater exploration ability than TRPO, so we choose K to be 2 in all environments for MP-PPO. Note that MPPO with different values of K all outperform the corresponding baseline policy optimization method (TRPO or PPO).

It is also worth noting that MPPO does not incur much more memory consumption for the population of policies in the policy

buffer, where it only uses 1.67% and 4.78% more memory for MP-TRPO ($K = 8$) and MP-PPO ($K = 2$) compared with TRPO and PPO respectively on SPARSEDOUBLEPENDULUM.
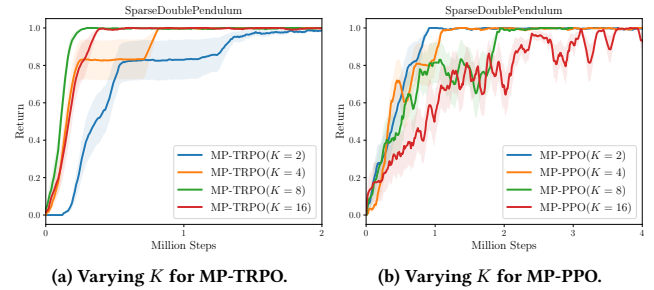

(a) Varying K for MP-TRPO. (b) Varying K for MP-PPO.

Figure 5: Ablation study of varying K.

4.3.2 The effect of the weight α. In the picking rule, α controls the trade-off between exploration and exploitation. A larger α emphasizes more on the exploration ability of the picked policy, but may fail to utilize the result of policy optimization. In addition, according to Theorem 3.1, a large α may lead to a temporary performance drop. On the other hand, a smaller α focuses more on exploiting the current best-performing policy in the policy buffer, where $\alpha = 0$ refers to always picking the best policy based on current estimation. We vary α for MP-TRPO on SPARSEDOUBLEPEN-DULUM, and the result is shown in Figure 6. As expected, a small $\alpha = 0.1$ achieves the best performance, so we fix α to be 0.1 in all environments for both MP-TRPO and MP-PPO.
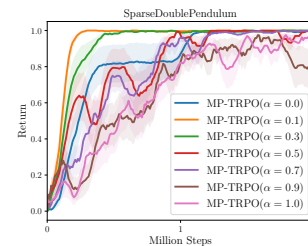

Figure 6: Ablation study of varying α.

4.4 Performance Comparison

Comparative analysis. The comparative results of MP-TRPO are demonstrated in Figure 7. As shown, MP-TRPO is consistently more sample efficient than Div-TRPO in all environments. In addition, it outperforms Curiosity-TRPO in all but one environment in terms of sample efficiency. The margin is larger especially in sparse environments. Table 1 summarizes the performance at the end of training, which shows that MP-TRPO achieves the best final performance in all environments.

Div-TRPO augments the loss function with a measure of distances between past policies. As trust-region methods limit the update, the distance among past policies is not large. Thus, the
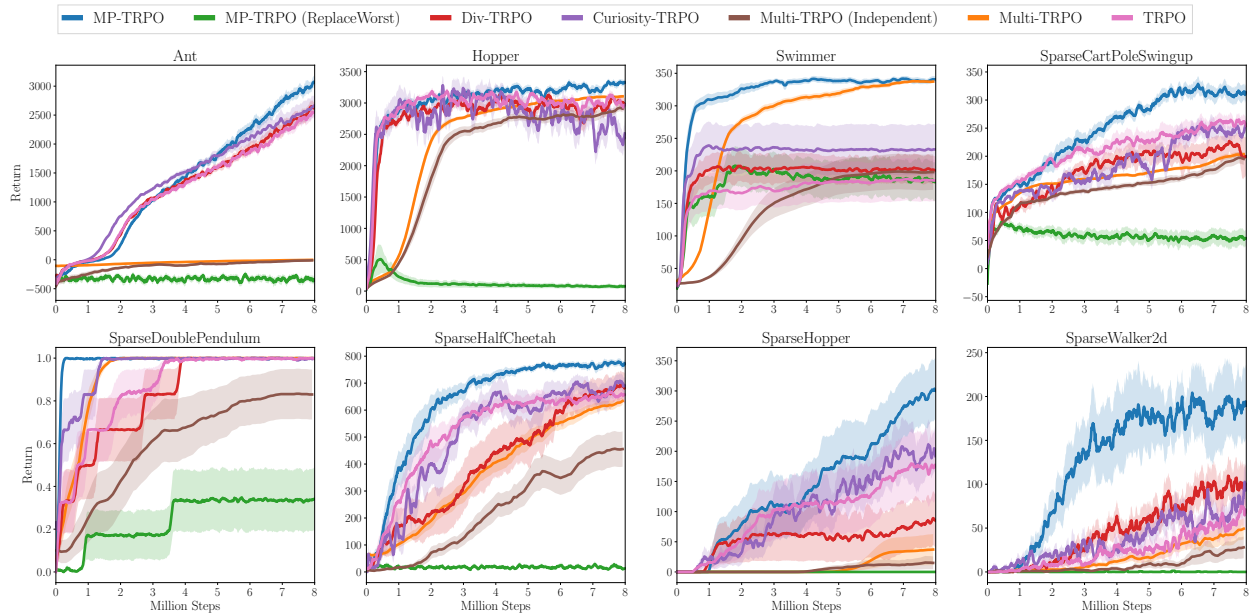
Figure 7: Performance comparison of MP-TRPO.

Table 1: Comparison of MP-TRPO on final performance (mean and confidence interval).

Environment	MP-TRPO	MP-TRPO (ReplaceWorst)	Div-TRPO	Curiosity-TRPO	Multi-TRPO (Independent)	Multi-TRPO	TRPO
ANT	**3017.02** (116.405)	-470.05 (57.65)	2635.00 (138.28)	2744.79 (190.23)	0.93 (1.99)	3.90 (1.19)	2558.33 (120.89)
HOPPER	**3257.55** (62.68)	76.25 (32.68)	2957.31 (82.94)	2786.99 (258.17)	2878.34 (91.57)	3138.41 (17.66)	2947.19 (111.43)
SWIMMER	**340.92** (4.39)	179.67 (29.63)	199.06 (20.58)	232.89 (38.93)	198.00 (27.47)	339.17 (2.21)	186.21 (32.63)
SPARSECARTPOLESWINGUP	**320.47** (14.48)	49.10 (16.99)	203.22 (33.56)	244.23 (11.42)	180.17 (11.50)	213.07 (4.07)	238.88 (11.08)
SPARSEDOUBLEPENDULUM	**1.00** (0.00)	0.33 (0.15)	**1.00** (0.00)	0.83 (0.12)	0.83 (0.01)	0.98 (0.01)	**1.00** (0.00)
SPARSEHALFCHEETAH	**756.02** (14.02)	24.52 (8.14)	699.03 (57.15)	656.98 (29.24)	438.50 (66.98)	676.22 (14.35)	639.32 (38.40)
SPARSEHOPPER	**302.32** (52.48)	0.00 (0.00)	89.60 (45.09)	188.47 (33.51)	14.63 (10.29)	38.33 (25.82)	182.63 (60.13)
SPARSEWALKER2D	**186.48** (43.15)	0.00 (0.00)	112.75 (33.29)	87.33 (25.18)	37.67 (12.57)	56.18 (17.27)	53.78 (15.57)

diversity-driven technique does not enable significant improvement of exploration on TRPO. Curiosity-TRPO augments the reward function with a curiosity term that measures how novel a state is. It encourages the agent to re-explore states that are known to be unfamiliar with. However, it can be challenging for the agent to first discover such states in sparse environments.

Effect of each component. Regarding the shared value network, Multi-TRPO outperforms Multi-TRPO (Independent), as it enables a better estimation of the value function. Additionally, MP-TRPO outperforms Multi-TRPO significantly, which validates that MP-TRPO enables efficient exploration with its mechanism to utilize the population of policies. As for the strategy for policy buffer updates, note that MP-TRPO (ReplaceWorst), where replacing the worst policy is a common strategy in evolutionary-based methods [22], performs poorly in all but one benchmark environments. After updating the picked policy, it replaces the worst-performing policy in the buffer with this improved policy. Under this updating scheme, the policy buffer loses the diversity of policies quickly and soon only stores K similar copies of a single policy. Thus, MP-TRPO (ReplaceWorst) performs worse than Multi-TRPO (Independent). In

contrast, the replacement strategy of MP-TRPO best preserves the diversity of the policy buffer while ensuring policy optimization.

Our results provide empirical evidence that MPPO is an efficient mechanism to fully utilize the population of policies without bringing high computation overhead.

Performance based on PPO. To show that MPPO is readily applicable to other baseline on-policy algorithms, we build it upon PPO, and evaluate the resulting MP-PPO algorithm by comparing it with the corresponding PPO, Multi-PPO, and Multi-PPO (Independent) algorithms. Learning curves are shown in Figure 8, with the final performance summarized in Table 2. Results show that MP-PPO outperforms the baseline methods in all environments in terms of sample efficiency and final performance, which demonstrates its effectiveness to encourage exploration.

5 RELATED WORK

In reinforcement learning, entropy is a critical term which relates to the uncertainty of the policy. Entropy-regularized reinforcement learning [14, 15, 27] optimizes the standard objective augmented by
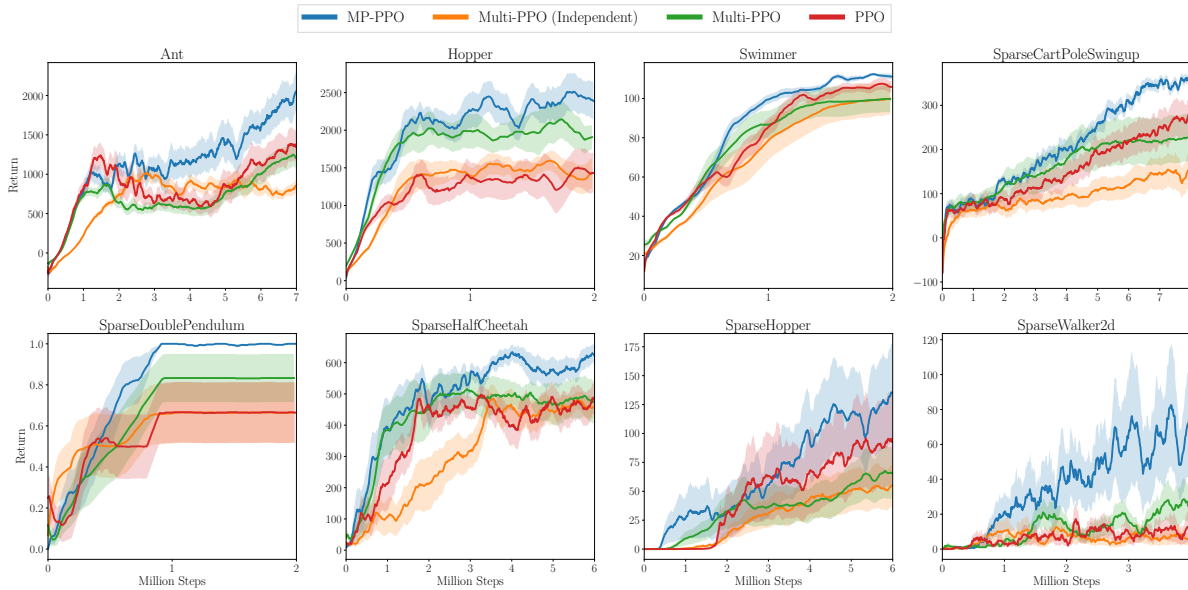
Figure 8: Performance comparison of MP-PPO.

Table 2: Comparison of MP-PPO on eventual performance (mean and confidence interval).

Environment	MP-PPO	Multi-PPO (Independent)	Multi-PPO	PPO
ANT	**1992.21** (216.33)	902.31 (101.00)	1131.90 (110.94)	1311.35 (220.93)
HOPPER	**2264.54** (234.95)	1449.33 (224.93)	2073.46 (161.47)	1457.18 (258.87)
SWIMMER	**109.96** (1.93)	100.28 (9.08)	98.79 (6.99)	106.12 (2.78)
SPARSECARTPOLESWINGUP	**352.12** (16.14)	135.65 (37.47)	238.02 (55.98)	294.88 (43.42)
SPARSEDOUBLEPENDULUM	**1.0**	0.67 (0.15)	0.83 (0.12)	0.67 (0.15)
SPARSEHALFCHEETAH	**593.37** (40.60)	456.65 (35.86)	478.70 (59.02)	463.07 (36.72)
SPARSEHOPPER	**131.35** (40.08)	57.88 (19.86)	63.75 (22.69)	90.40 (37.61)
SPARSEWALKER2D	**55.93** (31.98)	1.67 (1.17)	41.68 (19.91)	10.20 (7.18)

an entropy regularizer considering the distance with the random policy, and thus learns a stochastic policy for better exploration. Our method differs from them in that MPPO still optimizes the standard objective, where the picking rule involves the performance and an entropy bonus term.

Evolutionary methods have emerged to be an alternative of deep reinforcement learning [33, 39], and recent works that combine evolutionary methods and reinforcement learning [11, 22] have shown great power for better exploration and stability. There have also been a number of approaches improving exploration by combining evolutionary methods with deep reinforcement learning by maintaining a population of agents. Gangwani et al. [11] apply policy gradient methods to mutate the population. Khadka et al. [22] utilize a population of evolutionary actors to collect samples, where a reinforcement learning actor based on DDPG [24] is updated using these samples. Pourchot et al. [32] propose to combine the cross-entropy method and TD3 [9]. Our work differs from previous works in several aspects. First, we train a single policy at each iteration instead of a population of policies, and only the picked policy interacts with the environment. Second, we use multi-path to enable better exploration than single-path for on-policy algorithms, while previous works cannot be applied to on-policy algorithms.

Another approach related to our work is [43], where Zhang et al. propose to escape from local maxima for an off-policy algorithm, DDPG [24], by utilizing an ensemble of actors. The critic is updated according to the best action proposed by all actors that results in maximum Q-value, and all actors are trained in parallel. However, it cannot be applied to RL algorithms with stochastic policies.

6 CONCLUSION

We present Multi-Path Policy Optimization (MPPO), which uses a population of policies to improve exploration for on-policy reinforcement learning algorithms. We apply the MPPO method to TRPO and PPO, and show that the performance can be guaranteed during policy switching. We conduct extensive experiments on several MuJoCo tasks including environments with sparse rewards, and show that MPPO outperforms baselines significantly in both sample efficiency and final performance.

ACKNOWLEDGMENTS

REFERENCES

[1] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*. 1471–1479.

[2] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. 2018. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*. 8224–8234.

[3] Simyung Chang, John Yang, Jaeseok Choi, and Nojun Kwak. 2018. Genetic-gated networks for deep reinforcement learning. In *Advances in Neural Information Processing Systems*. 1747–1756.

[4] Yinlam Chow, Ofir Nachum, and Mohammad Ghavamzadeh. 2018. Path consistency learning in tsallis entropy regularized mdps. In *International Conference on Machine Learning*. 978–987.

[5] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2018. GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. In *International Conference on Machine Learning*. 1038–1047.

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*. 1329–1338.

[7] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. 2018. Noisy networks for exploration. In *International Conference on Learning Representations*.

[8] Justin Fu, John Co-Reyes, and Sergey Levine. 2017. Ex2: Exploration with exemplar models for deep reinforcement learning. In *Advances in Neural Information Processing Systems*. 2577–2587.

[9] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*. 1582–1591.

[10] Tanmay Gangwani, Qiang Liu, and Jian Peng. 2019. Learning self-imitating diverse policies. In *International Conference on Learning Representations*.

[11] Tanmay Gangwani and Jian Peng. 2017. Policy optimization by genetic distillation. In *International Conference on Learning Representations*.

[12] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. 2016. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247* (2016).

[13] Shixiang Shane Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. 2017. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in neural information processing systems*. 3846–3855.

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

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*. 1856–1865.

[16] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[17] Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, Tsu-Jui Fu, and Chun-Yi Lee. 2018. Diversity-Driven Exploration Strategy for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 10489–10500.

[18] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2016. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*. 1109–1117.

[19] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).

[20] Bingyi Kang, Zequn Jie, and Jiashi Feng. 2018. Policy optimization with demonstrations. In *International Conference on Machine Learning*. 2474–2483.

[21] Shauharda Khadka, Somdeb Majumdar, Santiago Miret, Evren Tumer, Tarek Nassar, Zach Dwiel, Yinyin Liu, and Kagan Tumer. 2019. Collaborative Evolutionary Reinforcement Learning. *arXiv preprint arXiv:1905.00976* (2019).

[22] Shauharda Khadka and Kagan Tumer. 2018. Evolution-Guided Policy Gradient in Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 1188–1200.

[23] Joel Z Leibo, Julien Perolat, Edward Hughes, Steven Wheelwright, Adam H Marblestone, Edgar Duéñez-Guzmán, Peter Sunehag, Iain Dunning, and Thore Graepel. 2019. Malthusian reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1099–1107.

[24] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*.

[25] Muhammad A Masood and Finale Doshi-Velez. 2019. Diversity-Inducing Policy Gradient: Using Maximum Mean Discrepancy to Find a Set of Diverse Policies. *arXiv preprint arXiv:1906.00088* (2019).

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

[27] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. 2018. Trust-pcl: An off-policy trust region method for continuous control. In *International Conference on Learning Representations*.

[28] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped DQN. In *Advances in neural information processing systems*. 4026–4034.

[29] Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. 2017. Count-based exploration with neural density models. In *International Conference on Machine Learning*. 2721–2730.

[30] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 16–17.

[31] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. 2018. Parameter space noise for exploration. In *International Conference on Learning Representations*.

[32] Aloïs Pourchot and Olivier Sigaud. 2019. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *International Conference on Learning Representations*.

[33] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[34] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[35] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*.

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

[37] Olivier Sigaud and Freek Stulp. 2019. Policy search in continuous action domains: an overview. *Neural Networks* (2019).

[38] William M Spears, Kenneth A De Jong, Thomas Bäck, David B Fogel, and Hugo De Garis. 1993. An overview of evolutionary computation. In *European Conference on Machine Learning*. Springer, 442–459.

[39] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).

[40] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. 2017. # Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*. 2753–2762.

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 5026–5033.

[42] G Zames, NM Ajlouni, NM Ajlouni, NM Ajlouni, JH Holland, WD Hills, and DE Goldberg. 1981. Genetic algorithms in search, optimization and machine learning. *Information Technology Journal* 3, 1 (1981), 301–302.

[43] Shangtong Zhang and Hengshuai Yao. 2019. Ace: An actor ensemble algorithm for continuous control with tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5789–5796.