Toward Practical Deep Reinforcement Learning: Sample-Efficient Self-Supervised Continuous Control

SUN, Hao

A thesis submitted in Partial Fullfillment
of the Requirements for the Degree of
Master of Philosophy
in
Information Engineering

The Chinese University of Hong Kong

May 2021

Abstract of thesis entitled: Toward Practical Deep Reinforcement Learning: Sample-Efficient Self-Supervised Continuous Control

Submitted by SUN, Hao

for the degree of Master of Philosophy in Information Engineering

at The Chinese University of Hong Kong in May 2021

Abstract

Solving goal-oriented tasks is an important but challenging problem in reinforcement learning (RL). For those tasks, the rewards are often sparse, making it difficult to learn a policy effectively. To tackle this difficulty, I first propose a new approach called *Policy Continuation with Hindsight Inverse Dynamics (PCHID)* in my MPhil study. This approach learns from Hindsight Inverse Dynamics based on Hindsight Experience Replay, enabling the learning process in a self-imitated manner and thus can be trained with supervised learning. Then the method is extended to multi-step settings with Policy Continuation. The proposed method is general, which can work in isolation or be combined with other on-policy and off-policy algorithms. On two multi-goal tasks GridWorld and FetchReach, PCHID significantly improves the sample efficiency as well as the final performance.

The second part of this thesis further extends PCHID. It provides a new formulation of GCRS tasks under the perspective of the drifted random walk in the state space. Based on the formulation we design a novel method called Evolutionary Stochastic Policy Distillation (ESPD) to solve GCRS tasks by reducing the First Hitting Time of the stochastic process. As a self-imitate approach, ESPD learns a target policy efficiently from a series of its stochastic-variant behaviors in a supervised learning manner. ESPD follows an evolutionary paradigm that applies perturbations upon the action of the target policy as the behavior policy for generating diverse experiences and then selects the superior experiences. Finally the target policy is updated by policy distillation on the selected experience transitions. The experiments on the OpenAI Fetch benchmarks show the improved sample efficiency of the proposed method.

Finally, this thesis extends the self-imitate learning paradigm into more general settings. Despite the remarkable progress made by the policy gradient algorithms in reinforcement learning (RL), sub-optimal policies usually result from the local exploration property of the policy gradient update. In the last part of this thesis, a method called Zeroth-Order Supervised Policy Improvement (ZOSPI) that exploits the estimated value function Q globally while preserving the local exploitation of the policy gradient methods is introduced. Experiments show that ZOSPI achieves competitive asymptotic performance on the MuJoCo benchmarks with a remarkable sample efficiency. Moreover, different from the conventional policy gradient methods, the policy learning of ZOSPI is conducted in a self-supervised manner. We show such a self-supervised learning paradigm has the flexibility of including optimistic exploration, adopting a non-parametric policy, and working with multi-modal policies.

摘要

在强化学习领域中，目标导向下的稀疏奖励任务是一个重要但极具挑战性的课题。在这些任务中，二值化的稀疏奖励使得策略的学习难以在传统算法下有效进行。为了解决这一问题，我在研究中首先提出了后视逆动力学模型下的策略延拓这一方法。这一方法基于后视经验回放这一先前工作，使得算法能够从失败的经验中学习如何成功达到后视目标。与先前工作不同的是，这一方法完全基于监督学习而非策略梯度。这一方法在策略延拓下随后被推广到了多步的情形从而对更复杂的任务进行处理。这一算法既可以单独作为学习算法工作也可以和其他强化学习方法结合来提升样本效率。在标准化的目标导向任务中进行的测试表明我们提出的这一方法大幅度提升了样本效率和收敛性能。

在论文的第二部分中，我们对第一部分提出的算法进行了进一步的改进。我们从带漂移项的随机游走角度重新定义了这类问题。基于这样的定义我们提出了随机策略蒸馏进化，该方法用减少首中时的方式对目标导向下的稀疏奖励任务进行学习。作为一种自我模仿的学习框架，此方法的策略从自己的随机扰动变种中找到更好的策略进行学习。我们在更困难的任务上标定了各种方法的性能，实验证明我们提出的这种方法进一步提升了样本效率。

尽管近年来策略梯度方法在强化学习任务中取得了较大的进展，由于这种方法只进行局部探索，得到的策略往往会收敛到局部极小值而非全局最优。论文的最后一部分将上述的自我模仿学习的强化学习框架推广到了更加一般的情况。我们将前文中用于解决目标导向问题的方法推广到一般的问题中，得到了零阶监督策略提升方法，这种方法基于零阶优化的思想，可以在全局进行探索和利用从而找到更优的策略。我们在多个强化学习任务中检验了所提出方法的性能，得到了当前最佳的样本效率和收敛策略性能。此外，由于此方法基于监督学习，我们通过非参数网络和多模态策略展示了方法的灵活性。

Contents

Chapter 1

Introduction

Goal-Conditioned Reward Sparse (GCRS) task is one of the challenging reinforcement learning tasks with extremely sparse rewards. In the task, the goal is combined with the current state as the policy input, while the agent is able to receive a positive reward only when the goal is achieved. In many cases, the GCRS task is also considered as the Multi-Goal task where the goal is not fixed and can be anywhere in the state space. Therefore the policy has to learn a general solution that can be applied to a set of similar tasks. For example, robotic object grasping is such a GCRS task: the target object could be anywhere on the table, the robot has to adjust its arm to reach the object and then grasp it. The objective of learning such a policy is to find a feasible path from the current state to the goal [73]. Similar tasks include the Multi-Goal benchmarks in robotics control [55].

In previous works, reward shaping [50], hierarchical reinforcement learning [23, 10], curriculum learning [11], and learning from demonstrations [61, 7, 5, 34, 49] were proposed to tackle the challenges of learning through sparse rewards. These approaches provide manual guidance from different perspectives. Besides, the Hindsight Experience Replay (HER) [39, 4] was proposed to relabel failed trajectories and assign hindsight credits as complementary to the primal sparse rewards, which is still a kind of Temporal Difference learning and relies on the value of reward.

In this work, we first improve the sample efficiency and stability of solving these GCRS tasks with an alternative approach based on supervised learning. For the first stage, he method of Policy Continuation with Hindsight Inverse Dynamics (PCHID) is proposed to learn with hindsight experiences in a supervised learning manner. Then we further improve the learning efficiency by synchronous learning. Specifically, by formulating the exploration in GCRS tasks as a random walk in the state space, solving the GCRS task is then equivalent to decreasing the first hitting time (FHT) in the random walk. The main idea is to encourage the policy producing trajectories that have shorter FHTs. With such a self-imitation manner, the policy learns to reach more and more *hindsight goals* [4] and becomes more knowledgeable to extrapolate its skills to solve the task. Based on this formulation, we further propose a new method for the GCRS tasks, which conforms a self-imitation learning approach and is independent of the rewards. Our agent learns from its own success or hindsight success, and extrapolates its knowledge to new situations, enabling the learning process to be executed in a much more efficient supervised learning manner.

Model-free Reinforcement Learning achieves great successes in many challenging tasks [48, 77, 53], however one hurdle for its application to real-world control problems is the insufficient sample efficiency. To improve the sample efficiency, off-policy methods [21, 29, 80, 45, 26] reuse the experiences generated by previous policies to optimize the current policy, therefore they can obtain a much higher sample efficiency than on-policy methods [66, 65]. Alternatively, SAC [31] increases sample efficiency by conducting more active exploration based on current policy, which is achieved by applying a maximum entropy framework [30] to the off-policy actor critic, and also results in the state-of-the-art asymptotic performance. OAC [18] further improves SAC by combining it with the Upper Confidence Bound heuristics [14] to motivate more informative exploration. Despite of their improvements, these methods all rely on a Gaussian policy and a local exploration strategy that simply adds noises to the action space, thus they might still converge to sub-optimal solutions as pointed out by tessler2019distributional.

In this work we aim to explore a new learning paradigm that can carry out non-local exploration as well as non-local exploitation in continuous control tasks for higher sample efficiency. Specifically, to better exploit the learned value function Q, we propose to search it globally for a better action, in contrast to previous attempts in policy gradient methods that only utilize its local information (e.g. the Jacobian matrix used in [68]). The idea behind our work is mostly related to the value-based policy gradient methods [45, 26], where the policy gradient optimization step takes the role of finding a well-performing action given a learned state-action value function. In previous work, the policy gradient step tackles the curse of dimensionality since it is intractable to directly search for the maximal value in the continuous action space [48].

To perform a global exploitation of the learned value function Q, we propose to apply a zeroth-order optimization scheme and update the target policy through supervised learning, which is inspired by works of evolution strategies [60, 19, 46] that adopt zeroth-order optimizations in the parameter space. Different from the standard policy gradient, combining the zeroth-order optimization with supervised learning forms a new way of policy update. Such an update avoids the local improvement of policy gradient: when policy gradient is applied, the target policy uses policy gradient to adjust its predictions according to the deterministic policy gradient theorem [68], but such adjustments can only lead to local improvements and may induce sub-optimal policies due to the non-convexity of the policy function [75]; on the contrary, the combination of zeroth-order optimization and supervised learning proposed in this paper can help the non-convex policy optimization to escape the local minimum as shown in our experiments.

Chapter 2

Related Work

2.1 Hindsight Experience Replay

Learning with sparse rewards in RL problems is always a leading challenge for the rewards are usually uneasy to reach with random explorations. Hindsight Experience Replay (HER) which relabels the failed rollouts as successful ones is proposed by Andrychowicz et al. [4] as a method to deal with such problem. The agent in HER receives a reward when reaching either the original goal or the relabeled goal in each episode by storing both original transition pairs s_t, g, a_t, r and relabeled transitions s_t, g', a_t, r' in the replay buffer. HER was later extended to work with demonstration data [49] and boosted with multi-processing training [55]. The work of Rauber et al. [58] further extended the hindsight knowledge into policy gradient methods using importance sampling.

2.2 Inverse Dynamics

Given a state transition pair (s_t, a_t, s_{t+1}), the inverse dynamics [38] takes (s_t, s_{t+1}) as the input and outputs the corresponding action a_t. Previous works used inverse dynamics to perform feature extraction [54, 16, 67] for policy network optimization. The actions stored in such transition pairs are always collected with a random policy so that it can barely be used to optimize the policy network directly. In our work, we use hindsight experience to revise the original transition pairs in inverse dynamics, and we call this approach Hindsight Inverse Dynamics. The details will be elucidated in the next section.

2.3 Auxiliary Task and Curiosity Driven Method

Mirowski et al. [47] propose to jointly learn the goal-driven reinforcement learning problems with an unsupervised depth prediction task and a self-supervised loop closure classification task, achieving data efficiency and task performance improvement. But their method requires extra supervision like depth input. Shelhamer et al. [67] introduce several self-supervised auxiliary tasks to perform feature extraction and adopt the learned features to reinforcement learning, improving the data efficiency and returns of end-to-end learning. Pathak et al. [54] propose to learn an intrinsic curiosity reward besides the normal extrinsic reward, formulated by prediction error of a visual feature space and improved the learning efficiency. Both of the approaches belong to self-supervision and utilize inverse dynamics during training. Although our method can be used as an auxiliary task and trained in self-supervised way, we improve the vanilla inverse dynamics with hindsight, which enables direct joint training of policy networks with temporal difference and self-supervised learning.

2.4 Learning with Experts and Policy Distillation.

Policy Distillation (PD) was proposed to extract the policy of a trained RL agent with a smaller network to improve the efficiency as well as the final performance or combine several task-specific agents together [59]. Latter extensions proposed to improve the learning efficiency [64], enhance multi-task learning [74, 6]. All of those methods start from a trained expert agent or human expert experience that can solve a specific task [20]. As a comparison, our proposed method focus on extracting knowledge from stochastic behaviors, which is capable to act as a feasible policy itself with regard to the primal task. In PD, a deterministic teacher policy is provided and can be queried to generate enough data to teach the student policy. However, in ESPD all the teacher policies are unknown as they are **stochastic variants** of the student policy. We regard those stochastic variants as they are from **unkonwn deterministic oracle policies** and distill those policies through the single corresponding trajectory. In ESPD, the teacher model is unknown and can not be queried as in PD.

2.5 Evolution Strategies and Parameter Noise.

The Evolution Strategy (ES) was proposed by [60] as an alternative to standard RL approaches, where the prevailing temporal difference based value function updates or policy gradient methods are replaced as perturbations on the parameter space to resemble the evolution. Later on, [17] improved the efficiency of ES by means of importance sampling. Besides, the method was also extended to be combined with Novelty-Seeking to further improve the performance [19]. Thereafter, [56] proposed to use Parameter Noise as an alternative to the action space noise injection for better exploration. They show such a perturbation on the parameter space can be not only used for ES methods, but also collected to improve the sample efficiency by combining it with traditional RL methods. While previous ES algorithms apply perturbations on the parameter noise and keep the best-performed variates, our approach implicitly execute the policy evolution by distilling better behaviors, therefore our approach can be regarded as an evolutionary method based on action space perturbation.

2.6 Supervised and Self-Imitate Approaches in RL.

Recently, several works put forward to use supervised learning to improve the stability and efficiency of RL. [82] propose to utilize supervised learning to tackle the overly large gradients problem in policy gradient methods. In order to improve sample efficiency, the work chose to first design a target distribution proposal and then used supervised learning to minimize the distance between present policy and the target policy distribution. The Upside-Down RL proposed by [63] used supervised learning to mapping states and rewards into action distributions, and therefore acted as a normal policy in RL. Their experiments show that the proposed UDRL method outperforms several baseline methods [69]. In the work of [70], a curriculum learning scheme is utilized to learn policies recursively. The self-imitation idea relevant to ESPD is also discussed in the concurrent work of [27], but ESPD further uses the SELECT function to improve the quality of collected data for self-imitation learning.

2.7 Policy Gradient Methods

The policy gradient methods solve the MDP by directly optimizing the policy to maximize the cumulative reward [81, 71]. While the prominent on-policy policy gradient methods like TRPO [66] and PPO [65] improve the learning stability via trust region updates, the off-policy methods such as DPG [68] and DDPG [45] can learn with a higher sample efficiency than the on-policy methods. The work of TD3 [26] further addresses the function approximation error and boosts the stability of DDPG with several improvements. Another line of works is the combination of policy gradient methods and the max-entropy principle, which leads to better exploration and stable asymptotic performances [30, 31]. All of these approaches adopt function approximators [72] for state or state-action value estimation as well as directionally uninformed Gaussian policies for policy parameterization, which lead to a local exploration behavior [18, 75].

2.8 Self-Supervised RL

Self-supervised learning or self-imitate learning is a rising stream as an alternative approach for model-free RL. Instead of applying policy gradient for policy improvement, methods of self-supervised RL update policies through supervised learning by minimizing the mean square error between target actions and current actions predicted by a policy network [70], or alternatively by maximizing the likelihood for stochastic policy parameterization [27]. While these works focus on the Goal-Conditioned tasks, in this work we aim at general RL tasks. Some other works use supervised learning to optimize the policy towards manually selected policies to achieve better training stability [79, 82, 1], but in our work the policy optimization is purely based on self-supervision with a much simpler formulation while achieving high performance.

2.9 Zeroth-Order Methods

Zeroth-order optimization methods, also called gradient-free methods, are widely used when gradients are difficult to compute. They approximate the local gradient with random samples around the current estimate. The works in wang2017stochastic,golovin2019gradientless show that a local zeroth-order optimization method has a convergence rate that depends logarithmically on the ambient dimension of the problem under some sparsity assumptions. It can also efficiently escape saddle points in non-convex optimizations [78, 9]. In RL, many studies have verified an improved sample efficiency of zeroth-order optimization [76, 46, 60]. In this work we provide a novel way of combining the local sampling and the global sampling to ensure that our algorithm approximates the gradient descent locally and is also able to find a better global region.

Chapter 3

Methods

3.1 Policy Continuation with Hindsight Inverse Dynamics

In this section we first briefly go through the preliminaries in Sec.3.1.1. In Sec.3.1.2 we retrospect a toy example introduced in HER as a motivating example. Sec.3.1.3 to 3.1.6 describe our method in detail.

3.1.1 Preliminaries

Markov Decision Process We consider a Markov Decision Process (MDP) denoted by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$, $\mathcal{A}$ are the finite state and action space, $\mathcal{P}$ describes the transition probability as $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. $r : \mathcal{S} \to \mathbb{R}$ is the reward function and $\gamma \in [0, 1]$ is the discount factor. $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ denotes a policy, and an optimal policy π^* satisfies $\pi^* = \arg\max_\pi \mathbb{E}_{s,a\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t)]$ where $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim \mathcal{P}(s_{t+1}|a_t, s_t)$ and an s_0 is given as a start state. When transition and policy are deterministic, $\pi^* = \arg\max_\pi \mathbb{E}_{s_0}[\sum_{t=0}^{\infty} \gamma^t r(s_t)]$ and $a_t = \pi(s_t)$, $s_{t+1} = \mathcal{T}(s_t, a_t)$, where $\pi : \mathcal{S} \to \mathcal{A}$ is deterministic and $\mathcal{T}$ models the deterministic transition dynamics. The expectation is over all the possible start states.

Universal Value Function Approximators and Multi-Goal RL The *Universal Value Function Approximator (UVFA)* [62] extends the state space of *Deep Q-Networks (DQN)* [48] to include goal state $g \in \mathcal{G}$ as part of the input, *i.e.*, s_t is extended to $(s_t, g) \in \mathcal{S} \times \mathcal{G}$. And the policy becomes $\pi : \mathcal{S} \times \mathcal{G} \to a_t$, which is pretty useful in the setting where there are multiple goals to achieve. Moreover, Schaul et al. [62] show that in such a setting, the learned policy can be generalized to previous unseen state-goal pairs. Our application of UVFA on Proximal Policy Optimization algorithm (PPO) [65] is straightforward. In the following of this work, we will use state-goal pairs to denote the *extended state space* $(s, g) \in \mathcal{S} \times \mathcal{G}$, $a_t = \pi(s_t, g)$ and $(s_{t+1}, g) = \mathcal{T}(s_t, a_t)$. The goal g is fixed within an episode, but changed across different episodes.

3.1.2 Revisiting the Bit-Flipping Problem

The bit-flipping problem was provided as a motivating example in HER [4], where there are n bits with the state space $\mathcal{S} = \{0, 1\}^n$ and the action space $\mathcal{A} = \{0, 1, ..0, n-1\}$. An action a corresponds

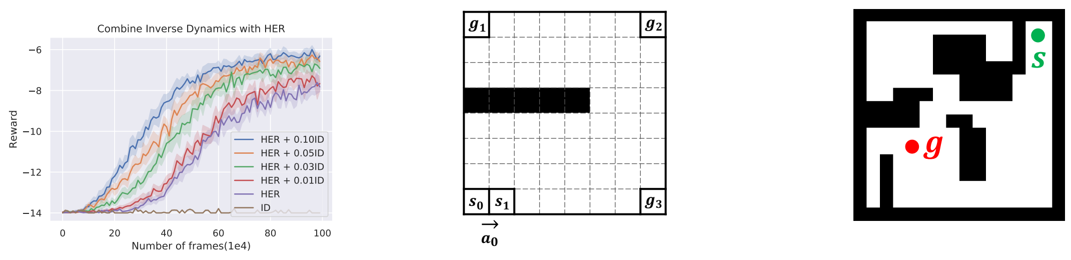


Figure 3.1: (a): Results in bit-flipping problem. (b): An illustration of flat state space. (c): An example of the GridWorld domain, which is a non-flat case.

to turn the a-th bit of the state. Each episode starts with a randomly generated state s_0 and a random goal state g. Only when the goal state g is reached the agent will receive a reward. HER proposed to relabel the failed trajectories to receive more reward signals thus enable the policy to learn from failures. However, the method is based on temporal difference thus the efficiency of data is limited. As we can learn from failures, here comes the question that can we learn a policy by supervised learning where the data is generated using hindsight experience?

Inspired by the self-imitate learning ability of human, we aim to employ self-imitation to learn how to get success in RL even when the original goal has not yet achieved. A straightforward way to utilize self-imitate learning is to adopt the inverse dynamics. However, in most cases the actions stored in inverse dynamics are irrelevant to the goals.

Specifically, transition tuples like $((s_t, g), (s_{t+1}, g), a_t)$ are saved to learn the inverse dynamics of goal-oriented tasks. Then the learning process can be executed simply as classification when action space is discrete or regression when action space is continuous. Given a neural network parameterized by ϕ, the objective of learning inverse dynamics is as follows,

$$\phi = \arg\min_{\phi} \sum_{s_t, s_{t+1}, a_t} ||f_\phi((s_t, g), (s_{t+1}, g)) - a_t||^2. \tag{3.1}$$

Due to the unawareness of the goals while the agent is taking actions, the goals g in Eq.(3.1) are only placeholders. Thus, it will cost nothing to replace g with $g' = m(s_{t+1})$ but result in a more meaningful form, *i.e.*, encoding the following state as a hindsight goal. That is to say, if the agent wants to reach g' from s_t, it should take the action of a_t, thus the decision making process is aware of the hindsight goal. We adopt f_ϕ trained from Eq.(3.1) as an additional module incorporating with HER in the Bit-flipping environment, by simply adding up their logit outputs. As shown in Fig.3.1(a), such an additional module leads to significant improvement. We attribute this success to the flatness of the state space. Fig.3.1(b) illustrates such a flatness case where an agent in a grid map is required to reach the goal g_3 starting from s_0: if the agent has already known how to reach s_1 in the east, intuitively, it has no problem to extrapolate its policy to reach g_3 in the farther east.

Nevertheless, success is not always within an effortless single step reach. Reaching the goals of g_1 and g_2 are relatively harder tasks, and navigating from the start point to goal point in the GridWorld domain shown in Fig.3.1(c) is even more challenging. To further employ the self-imitate learning and overcome the single step limitation of inverse dynamics, we come up with a new approach called Policy Continuation with Hindsight Inverse Dynamics.

3.1.3 Perspective of Policy Continuation on Multi-Goal RL Task

Our approach is mainly based on policy continuation over sub-policies, which can be viewed as an emendation of the spontaneous extrapolation in the bit-flipping case.

Definition 1: Policy Continuation(PC) *Suppose π is a policy function defined on a non-empty sub-state-space $\mathcal{S}_U$ of the state space $\mathcal{S}$, i.e., $\mathcal{S}_U \subset \mathcal{S}$. If $\mathcal{S}_V$ is a larger subset of $\mathcal{S}$, containing $\mathcal{S}_U$, i.e., $\mathcal{S}_U \subset \mathcal{S}_V$ and Π is a policy function defined on $\mathcal{S}_V$ such that*

$$\Pi(s) = \pi(s) \qquad \forall s \in \mathcal{S}_U$$

then we call Π a policy continuation of π, or we can say the restriction of Π to $\mathcal{S}_U$ is the policy function π.

Denote the optimal policy as $\pi^* : (s_t, g_t) \to a_t$, we introduce the concept of k-step solvability:

Definition 2: k-Step Solvability *Given a state-goal pair (s, g) as a task of a certain system with deterministic dynamics, if reaching the goal g needs at least k steps under the optimal policy π^* starting from s, i.e., starting from $s_0 = s$ and execute $a_i = \pi^*(s_i, g)$ for $i = \{0, 1, ..., k-1\}$, the state $s_k = \mathcal{T}(s_{k-1}, a_{k-1})$ satisfies $m(s_k) = g$, we call the pair (s, g) has k-step solvability, or (s, g) is k-step solvable.*

Ideally the k-step solvability means the number of steps it should take from s to g, given the maximum permitted action value. In practice the k-step solvability is an evolving concept that can gradually change during the learning process, thus is defined as "whether it can be solve with π_{k-1} within k steps after the convergence of π_{k-1} trained on (k-1)-step HIDs".

We follow HER to assume a mapping $m : \mathcal{S} \to \mathcal{G}$ s.t. $\forall s \in \mathcal{S}$ the reward function $r(s, m(s)) = 1$, thus, the information of a goal g is encoded in state s. For the simplest case we have m as identical mapping and $\mathcal{G} = \mathcal{S}$ where the goal g is considered as a certain state s of the system.

Following the idea of recursion in curriculum learning, we can divide the finite state-goal space into $T + 2$ parts according to their k-step solvability,

$$\mathcal{S} \times \mathcal{G} = (\mathcal{S} \times \mathcal{G})_0 \cup (\mathcal{S} \times \mathcal{G})_1 \cup ... \cup (\mathcal{S} \times \mathcal{G})_T \cup (\mathcal{S} \times \mathcal{G})_U \tag{3.2}$$

where $(s, g) \in \mathcal{S} \times \mathcal{G}$, T is a finite time-step horizon that we suppose the task should be solved within, and $(\mathcal{S} \times \mathcal{G})_i, i \in \{0, 1, 2, ...T\}$ denotes the set of i-step solvable state-goal pairs, $(s, g) \in (\mathcal{S} \times \mathcal{G})_U$ denotes unsolvable state-goal pairs, *i.e.*, (s, g) is not k-step solvable for $\forall k \in \{0, 1, 2, ..., T\}$, and $(\mathcal{S} \times \mathcal{G})_0$ is the trivial case $g = m(s_0)$. As the optimal policy only aims to solve the solvable state-goal pairs, we can take $(\mathcal{S} \times \mathcal{G})_U$ out of consideration. It is clear that we can define a disjoint sub-state-goal space union for the solvable state-goal pairs

Definition 3: Solvable State-Goal Space Partition *Given a certain environment, any solvable state-goal pairs can be categorized into only one sub state-goal space by the following partition*

$$\mathcal{S} \times \mathcal{G} \backslash (\mathcal{S} \times \mathcal{G})_U = \bigcup_{j=0}^{T} (\mathcal{S} \times \mathcal{G})_j \tag{3.3}$$

Then, we define a set of sub-policies $\{\pi_i\}, i \in \{0, 1, 2, ..., T\}$ on solvable sub-state-goal space $\bigcup_{j=0}^{i}(\mathcal{S} \times \mathcal{G})_j$ respectively, with the following definition

Definition 4: Sub Policy on Sub Space *π_i is a sub-policy defined on the sub-state-goal space $(\mathcal{S} \times \mathcal{G})_i$. We say π_i^* is an optimal sub-policy if it is able to solve all i-step solvable state-goal pair tasks in i steps.*

Corollary 1: *If $\{\pi_i^*\}$ is restricted as a policy continuation of $\{\pi_{i-1}^*\}$ for $\forall i \in \{1, 2, ...k\}$, π_i^* is able to solve any i-step solvable problem for $i \leq k$. By definition, the optimal policy π^* is a policy continuation of the sub policy π_T^*, and π_T^* is already a substitute for the optimal policy π^*.*

We can recursively approximate π^* by expanding the domain of sub-state-goal space in policy continuation from an optimal sub-policy π_0^*. While in practice, we use neural networks to approximate such sub-policies to do policy continuation. We propose to parameterize a policy function $\pi = f_\theta$ by θ with neural networks and optimize f_θ by self-supervised learning with the data collected by Hindsight Inverse Dynamics (HID) recursively and optimize π_i by joint optimization.

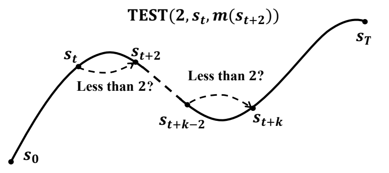
3.1.4 Hindsight Inverse Dynamics

One-Step Hindsight Inverse Dynamics One step HID data can be collected easily. With n randomly rollout trajectories $\{(s_0, g), a_0, r_0, (s_1, g), a_1, ..., (s_T, g), a_T, r_T\}_i, i \in \{1, 2, ..., n\}$, we can use a modified inverse dynamics by substituting the original goal g with hindsight goal $g' = m(s_{t+1})$ for every s_t and result in $\{(s_0, m(s_1)), a_0, (s_1, m(s_2)), a_1, ..., (s_{T-1}, m(s_T)), a_{T-1}\}_i, i \in \{1, 2, ..., n\}$. We can then fit f_{θ_1} by

$$\theta_1 = \arg\min_\theta \sum_{s_t, s_{t+1}, a_t} ||f_\theta((s_t, m(s_{t+1})), (s_{t+1}, m(s_{t+1}))) - a_t||^2 \tag{3.4}$$

By collecting enough trajectories, we can optimize f_θ implemented by neural networks with stochastic gradient descent [13]. When m is an identical mapping, the function f_{θ_1} is a good enough approximator for π_1^*, which is guaranteed by the approximation ability of neural networks [33, 35, 43]. Otherwise, we should adapt Eq. 3.4 as $\theta_1 = \arg\min_\theta \sum_{s_t, s_{t+1}, a_t} ||f_\theta((s_t, m(s_{t+1})), m(s_{t+1})) - a_t||^2$, *i.e.*, we should omit the state information in future state s_{t+1}, to regard f_{θ_1} as a policy. And in practice it becomes $\theta_1 = \arg\min_\theta \sum_{s_t, s_{t+1}, a_t} ||f_\theta(s_t, m(s_{t+1})) - a_t||^2$.

Multi-Step Hindsight Inverse Dynamics Once we have $f_{\theta_{k-1}}$, an approximator of π_{k-1}^*, k-step HID is ready to get. We can collect valid k-step HID data recursively by testing whether the k-step HID state-goal pairs indeed need k steps to solve, i.e., for any k-step transitions $\{(s_t, g), a_t, r_t, ..., (s_{t+k}, g), a_{t+k}, r_{t+k}\}$, if our policy π_{k-1}^* at hand can not provide with another solution from $(s_t, m(s_{t+k}))$ to $(s_{t+k}, m(s_{t+k}))$ in less than k steps, the state-goal pair $(s_t, m(s_{t+k}))$ must be k-step solvable, and this pair together with the action a_t will be marked as $(s_t^{(k)}, m(s_{t+k}^{(k)})), a_t^{(k)}$. Fig.3.2 illustrates this process. The testing process is based on a function TEST$(\cdot)$ and we will focus on the selection of TEST in Sec.3.1.6. Transition pairs like this will be collected to optimize θ_k. In practice, we leverage joint training to



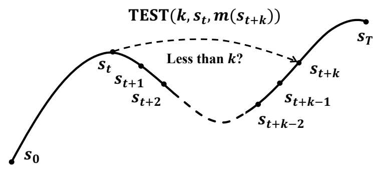


Figure 3.2: Test whether the transitions are 2-step (left) or k-step (right) solvable. The TEST function returns True if the transition $s_t \to s_{t+k}$ needs at least k steps.

ensure f_{θ_k} to be a policy continuation of $\pi_i^*, i \in \{1, ..., k\}$ i.e.,

$$\theta_k = \arg\min_\theta \sum_{s_t^{(i)}, s_{t+i}^{(i)}, a_t^{(i)}, i \in \{1,...,k\}} ||f_\theta((s_t, m(s_{t+i})), (s_{t+i}, m(s_{t+i}))) - a_t||^2 \tag{3.5}$$

3.1.5 Dynamic Programming Formulation

For most goal-oriented tasks, the learning objective is to find a policy to reach the goal as soon as possible. In such circumstances,

$$L^\pi(s_t, g) = L^\pi(s_{t+1}, g) + 1 \tag{3.6}$$

where $L^\pi(s, g)$ here is defined as the number of steps to be executed from s to g with policy π and 1 is the additional

$$L^{\pi^*}(s_t, g) = L^{\pi^*}(s_{t+1}, g) + 1 \tag{3.7}$$

and $\pi^* = \arg\min_\pi L^\pi(s_t, g)$ As for the learning process, it is impossible to enumerate all possible intermediate state s_{t+1} in the continuous state space and

Suppose now we have the optimal sub-policy π_{k-1}^* of all i-step solvable problems $\forall i \le k-1$, we will have

$$L^{\pi_k^*}(s_t, g) = L^{\pi_{k-1}^*}(s_{t+1}, g) + 1 \tag{3.8}$$

holds for any $(s_t, g) \in (\mathcal{S} \times \mathcal{G})_k$. We can sample trajectories by random rollout or any unbiased policies and choose some feasible (s_t, g) pairs from them, *i.e.*, any s_t and s_{t+k} in a trajectory that can not be solved by the

$$s_t \xrightarrow{a_t = \pi_k(s_t, s_{t+k})} s_{t+1} \xrightarrow{\pi_{k-1}^*} s_{t+k} \tag{3.9}$$

Such a recursive approach starts from π_1^*, which can be easily approximated by trained with self supervised learning by any given (s_t, s_{t+1}) pairs for $(s_t, s_{t+1}) \in (\mathcal{S} \times \mathcal{G})_1$ by definition.

The combination of PC and with multi-step HID leads to our algorithm PCHID. PCHID can work alone or as an auxiliary module with other RL algorithms. We discuss three different combination methods of PCHID and other algorithms in Sec.4.1.3. The full algorithm of the PCHID is presented as Algorithm 1.

Algorithm 1 Policy Continuation with Hindsight Inverse Dynamics (PCHID)

```
Require policy $\pi_b(s, g)$, reward function $r(s, g)$ (equal to 1 if $g = m(s)$ else 0), a buffer for PCHID
$\mathbb{B} = \{\mathbb{B}_1, \mathbb{B}_2, ..., \mathbb{B}_{T-1}\}$, a list $\mathbb{K}$
Initialize $\pi_b(s, g)$, $\mathbb{B}$, $\mathbb{K} = [1]$
for episode $= 1, M$ do
  generate $s_0$, $g$ by the system
  for $t = 0, T - 1$ do
    Select an action by the behavior policy $a_t = \pi_b(s_t, g)$
    Execute the action $a_t$ and get the next state $s_{t+1}$
    Store the transition $((s_t, g), a_t, (s_{t+1}, g))$ in a temporary episode buffer
  end for
  for $t = 0, T - 1$ do
    for $k \in \mathbb{K}$ do
      calculate additional goal according to $s_{t+k}$ by $g' = m(s_{t+k})$
      if TEST$(k, s_t, g')$ = True then
        Store $(s_t, g', a_t)$ in $\mathbb{B}_k$
      end if
    end for
  end for
  Sample a minibatch $B$ from buffer $\mathbb{B}$
  Optimize behavior policy $\pi_b(s_t, g')$ to predict $a_t$ by supervised learning
  if Converge then
    Add max$(\mathbb{K}) + 1$ in $\mathbb{K}$
  end if
end for
```

3.1.6 On the Selection of TEST Function

In Algorithm 1, a crucial step to extend the $(k-1)$-step sub policy to k-step sub policy is to test whether a k-step transition $s_t \to s_{t+k}$ in a trajectory is indeed a k-step solvable problem if we regard s_t as a start state s_0 and $m(s_{t+k})$ as a goal g. We propose two approaches and evaluate both in Sec.4.1.

Interaction A straightforward idea is to reset the environment to s_t and execute action a_t by policy π_{k-1}, followed by execution of $a_{t+1}, a_{t+2}, ...$, and record if it achieves the goal in less than k steps. We call this approach *Interaction* for it requires the environment to be resettable and interact with the environment. This approach can be portable when the transition dynamics is known or can be approximated without a heavy computation expense.

Random Network Distillation (RND) Given a state as input, the RND [15] is proposed to provide exploration bonus by comparing the output difference between a fixed randomly initialized neural network N_A and another neural network N_B, which is trained to minimize the output difference between N_A and N_B with previous states. After training N_B with $1, 2, ..., k-1$ step transition pairs to minimize the output difference between N_A and N_B, since N_B has never seen k-step solvable transition pairs, these pairs will be differentiated for they lead to larger output differences.

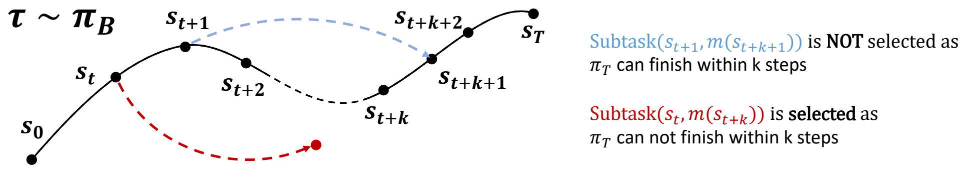

$\tau \sim \pi_B$
s_0
s_t
s_{t+1}
s_{t+2}
s_{t+k}
s_{t+k+1}
s_{t+k+2}
s_T
Subtask($s_{t+1}, m(s_{t+k+1})$) is NOT selected as π_T can finish within k steps
Subtask($s_t, m(s_{t+k})$) is selected as π_T can not finish within k steps

Figure 3.3: Illustration of the selection process: first we generate episodes with the stochastic behavior policy π_B, which is composed of the deterministic target policy π_T and a noise term drawn from Gaussian, then we check the superiority of generated transitions over the target policy. If π_T can not find a shorter path for a transition generated by π_B, the select function will return True and the transition will be stored for Stochastic Policy Distillation. Therefore, π_T will learn to evolve to solve more sub-task, , transitions, continuously.

3.1.7 Synchronous Improvement

In PCHID, the learning scheme is set to be curriculum, *i.e.*, the agent must learn to master easy skills before learning complex ones. However, in general the efficiency of finding a transition sequence that is i-step solvable decreases as i increases. The size of buffer $\mathbb{B}_i$ is thus decreasing for $i = 1, 2, 3, ..., T$ and the learning of π_i might be restricted due to limited experiences. Besides, in continuous control tasks, the k-step solvability means the number of steps it should take from s to g, given the maximum permitted action value. In practice the k-step solvability can be treated as an evolving concept that changes gradually as the learning goes. Specifically, at the beginning, an agent can only walk with small paces as it has learned from experiences collected by random movements. As the training continues, the agent is confident to move with larger paces, which may change the distribution of selected actions. Consequently, previous k-step solvable state goal pairs may be solved in less than k steps.

Based on the efficiency limitation and the progressive definition of k-step solvatbility, we propose a synchronous version of PCHID. This lead to the second part of this thesis: Evolutionary Stochastic Policy Distillation.

3.2 Evolutionary Stochastic Policy Distillation

3.2.1 Problem Re-Formulation

In a given goal-oriented reinforcement learning task, we assume there exists an unknown metric $\mathcal{D}(s_t, g)$ that represents the distance between the current state s_t and the desired goal state g. For example $\mathcal{D}(s, g)$ is the Euclidean distance in barrier-free navigation tasks; or the Manhattan distance in navigation tasks with obstacles.

A feasible solution of the task should be a policy $\pi(s_t, g)$ that outputs an action a_t, such that the distance $\mathcal{D}(s_{t+1}, g) \le \mathcal{D}(s_t, g)$ for deterministic dynamics, or $\mathbb{E}[\mathcal{D}(s_{t+1}, g)] \le \mathbb{E}[\mathcal{D}(s_t, g)]$ for stochastic dynamics. We assume $\mathcal{D}(s, g)$ is continuous and differentiable on s, and $\mathrm{d}s = -\xi \frac{\partial \mathcal{D}(s,g)}{\partial s}\mathrm{d}t$ is a feasible move, as it decreases the distance between s and g when ξ is sufficiently small. We further assume the state is a vector, the state transition $\Delta s_t = s_{t+1} - s_t$ is determined by both the dynamics $\phi_s(\cdot) : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ and the action $a_t = \pi(s_t, g)$ provided by the policy. We may write a sufficient condition for a feasible policy:

$$\phi_s(\pi(s,g)) = -\xi \frac{\partial \mathcal{D}(s,g)}{\partial s} \mathrm{d}t, \tag{3.10}$$

we further assume $\phi_s^{-1}(\cdot)$ exists[2], $\forall a \in \mathcal{A}$, $\phi_s^{-1}(\phi_s(a)) = a$. Hence, by parameterizing the policy $\pi(s,g)$ with θ, we have

$$\pi_\theta(s,g) = \phi_s^{-1}\left(-\xi \frac{\partial \mathcal{D}(s,g)}{\partial s} \mathrm{d}t\right), \tag{3.11}$$

is a feasible policy, , it tends to solve the GCRS task as it continuously minimizes the distance between the current state and the goal. The above equation tells us, in order to learn a well-performing policy, the policy should learn two unknown functions: the inverse dynamics $\phi_s^{-1}(\cdot)$ and the derivatives of distance metric $\mathcal{D}(\cdot,\cdot)$ over s and g with regard to the state s. In PCHID, Hindsight Inverse Dynamics is used as a practical policy learning method in such GCRS tasks. Specifically, in Inverse Dynamics, a model parameterized by θ is optimized by minimizing the mean square error of predicted action $\hat{a}_t$ and executed action a_t between adjacent states s_t and s_{t+1}, $\theta = \arg\min_\theta (a_t - \hat{a}_t)^2 = \arg\min_\theta (a_t - \pi_\theta(s_t, s_{t+1}))^2$. The HID revises the latter s_{t+1} with its goal correspondence $g_{t+1} = m(s_{t+1})$, where the mapping m is assumed to be known in normal GCRS task settings. $m : \mathcal{S} \to \mathcal{G}$ s.t. $\forall s \in \mathcal{S}$ the reward function $r(s, m(s)) = 1$. In the single step transition setting, the learning objective of the policy is to learn HID by

$$\theta = \arg\min_\theta (a_t - \hat{a}_t)^2 = \arg\min_\theta (a_t - \pi_\theta(s_t, g_{t+1}))^2. \tag{3.12}$$

Eq.3.12 shows the HID can be used to train a policy with supervised learning by minimizing the prediction error. However, to get more capable policy that is able to solve harder cases, training the policy only with 1-step HID is not enough. The work of PCHID then proposed to check the optimality of multi-step transitions with a TEST function and learn multi-step HID recursively. Such an explicit curriculum learning strategy is not efficient as multi-step transitions can only be collected after the convergence of previous sub-policies.

Here we interpret how PCHID works from the SDE perspective. Practically, a policy is always parameterized as a neural network trained from scratch to solve a given task. At beginning the policy will not be fully goal-oriented as a feasible policy. With random initialization, the policy network will just perform random actions regardless of what state and goal are taken as inputs. We use a coefficient ϵ to model the goal awareness of the policy, e.g., $\epsilon = 0$ denotes a purely random policy, and $\epsilon = 1$ denotes a better policy. In order to collect diverse experiences and improve our target policy, we follow traditional RL approaches to assume a random noise term denoted by N with coefficient σ to execute exploration. Hence, the behavioral policy becomes:

$$\pi_{\text{behave}} = \pi_\theta(s,g) + \sigma N = \epsilon \phi_s^{-1}\left(-\xi \frac{\partial \mathcal{D}(s,g)}{\partial s} \mathrm{d}t\right) + \sigma N. \tag{3.13}$$

The behavioral policy above combines a deterministic term and a stochastic term, which in practice can be implemented as a Gaussian noise or OU-noise [45]. Although we assume a deterministic policy $\pi_\theta(s,g)$ here, the extension to stochastic policies is straightforward, e.g., the network can

[2]The assumption can be released to a existence of pseudo inverse: $\phi_s^{-1}(\phi_s(a)) \in A'$, where A' is a set s.t. $\forall a' \in A'$, $\phi_s(a') = \phi_s(a)$.

predict the mean value and the standard deviation of an action to form a Gaussian policy family and the Mixture Density Networks [12] can be used for more powerful policy representations.

With such a formulation, the PCHID can be regarded as a method that *explicitly* learns the inverse dynamics with HID, and progressively learns the metric $\mathcal{D}(s,g)$ with Policy Continuation (PC). In this work, we justify that the approach can be extended to a more efficient synchronous setting that *implicitly* learns the inverse dynamics $\phi_s^{-1}(\cdot)$ and the derivatives of distance metric $\mathcal{D}(\cdot,\cdot)$ with regard to state s at the same time. The key insight of our proposed method is *minimizing the First Hitting Time [3] of a drifted random walk* (Eq.3.13).

Concretely, the simplest case of Eq.3.13 is navigating in the Euclidean space, where the distance metric is $\mathcal{D}(s,g) = ||g-s||_2$ and the transition dynamics is an identical mapping, $\phi_s(a) = a \in \mathcal{A} \subset \mathcal{S}$, and by applying a Gaussian noise on the action space, we have

$$\pi(s,g) = ds = \epsilon \frac{g-s}{||g-s||_2} \mathrm{d}t + \sigma \mathrm{d}W_t, \tag{3.14}$$

which is a Stochastic Differential Equation (SDE). As our learning objective is to increase the possibility of reaching the goal in a finite time horizon, the problem can be formulated as minimizing the FHT $\tau = \inf\{t > 0 : s_t = g | s_0 > g\}$, hitting the goal in the state space. In practice, the goal state is always a region in the state space [55], and therefore the task is to cross the region as soon as possible.

3.2.2 Practical Implementation

As we analyzed in the previous SDE formulation, the learning objective is to minimize FHT, providing the advantage of learning k-step transitions **simutaneously**. This advantage in principle can not be used in PCHID as its PC step requires implicit curriculum to guarantee the optimality of sub-policies.

In this section, we propose a practical algorithm that combines evolutionary strategy with policy distillation to minimize FHT. Specifically, ESPD maintains a target deterministic policy π_T, parameterized as a policy network, and a behavioral stochastic policy π_B

$$\pi_B = \pi_T + \tilde{\eta}, \tilde{\eta} \sim \mathcal{N}(0, \sigma^2), \tag{3.15}$$

according to Eq.3.13, the behavior policy comes from adding a Gaussian exploration noise upon the target policy π_T, as in previous deterministic policy learning literature [45, 26]. For the policy update step, ESPD use the evolutionary idea by distilling the well-performed behavior policies, in terms of FHT, to the target policy, instead of applying policy gradient or the zeroth-order approximation of policy gradient [60] to the target policy. Concretely, during training, π_B first interacts with the environment and collects a batch of transition samples, permitting us to generate a batch of HIDs, regardless of their optimality. These HIDs contain a set of transition tuples (s, g', a), where g' denotes the hindsight goal. , the starting point, final achieved goal, and the corresponding action are included in each of these transition tuples. From an oracle-perspective, these HIDs can be regarded as generated from a series of *unknown deterministic policies* instead of a known stochastic policy π_B, each provides a individual solution for the state-goal pair task (s, g'). Among these unknown oracle-policies, some are better than our current target policy π_T in terms of FHT, which means they are

Algorithm 2 ESPD

```
Require
  1.a reward function $r(s, g) = 1$ if $g = m(s)$ else 0
  2.a Horizon list $\mathcal{K} = [1, 2, ..., K]$
Initialize target policy $\pi_T(s, g)$, $\pi_B(s, g) = \pi_T(s, g) + \tilde{\eta}, \tilde{\eta} \sim \mathcal{N}(0, \sigma^2)$, replay buffer $\mathcal{B} = \{\}$
for episode $= 1, M$ do
  Generate $s_0$, $g$ by the environment
  for $t = 1, T$ do
    Select an action by the behavior policy $a_t = \pi_B(s_t, g)$
    Execute the action $a_t$ and get the next state $s_{t+1}$
  end for
  for $t = 1, T$ do
    for $k = 1, K$ do
      Calculate additional goal according to $s_{t+k}$ by $g' = m(s_{t+k})$
      if SELECT$(s_t, g')$ = True then
        Store $(s_t, g', a_t)$ in $\mathcal{B}$
      end if
    end for
  end for
  Sample a minibatch $B$ from buffer $\mathcal{B}$
  Optimize target policy $\pi_T(s_t, g')$ according to Eq.3.16
  Update behavior policy $\pi_B = \pi_T + \tilde{\eta}, \tilde{\eta} \sim \mathcal{N}(0, \sigma^2)$
end for
```

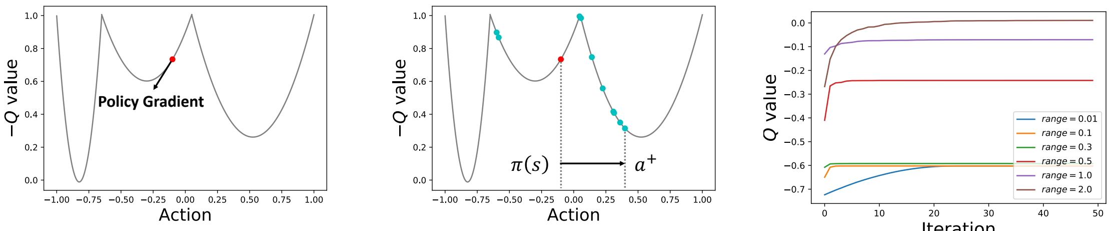
able to solve the certain state-goal pair task in fewer steps, or they are able to solve some sub-tasks while the π_T is not. Although we are not able to access these well-performing oracle-policies directly, we can distill the useful knowledge from them to π_T through their corresponding HIDs.

In practice, we use a SELECT function to distinguish those HIDs that outperform π_T and store them in a buffer $\mathcal{B} = \{(s_i, g'_i, a_i)\}_{i=1,2,...,}$. The SELECT function can be implemented in different ways, (1) reset the environment to a given previous state, which is always tractable in simulation [49], (2) use classifiers, dynamic models or heuristics [70]. In this work we adopt (1) and leave the usage of model-based SELECT functions to the future work. To implement (1), the SELECT function takes in an episode generated by π_B. Suppose the episode $(s_t, a_t, s_{t+1}, a_{t+1}, ..., s_{t+k})$ is of length k, the SELECT function resets environment to the starting state of this episode s_t and runs π_T for up to k steps, trying to reach the final achieved state s_{t+k}. , at every step, an action of $\pi_T(s, m(s_{t+k}))$ is performed. If π_T is **NOT** able to reach s_{t+k} within k steps, the corresponding transition tuple $(s_t, m(s_{t+k}), a_t)$ will be collected in the buffer $\mathcal{B}$ and π_T will learn from these tuples later. The procedure is illustrated in Fig.3.3.

Then, we can apply Stochastic Policy Distillation (SPD) to distill the knowledge from the well-performing oracle-policies to π_T so that π_T may evolve to be more capable to tackle the same sub-tasks. To be specific, we use supervised learning to minimize the difference between the action stored in the HID buffer and the action π_T predicted. The SPD is conducted as

$$\pi_T = \arg\min_{\pi_T} \frac{1}{N} \sum_{i=1}^{N} (\pi_T(s_i, g'_i) - a_i)^2, \tag{3.16}$$

where (s_i, g'_i, a_i) are sampled from the HID buffer $\mathcal{B}$. From this point of view, the ESPD method is composed of evolution strategy and policy distillation, where a stochastic behavior policy π_B acts as



(a) Policy Gradient (b) ZOSPI (c) Simulation

Figure 3.4: (a) Landscape of Q value for a 1-dim continuous control task. Policy gradient methods optimize the policy according to the local information of Q. (b) For the same task, ZOSPI directly updates the predicted actions to the sampled action with the largest Q value. (c) Simulation results, where in each optimization iteration 10 actions are uniformly sampled under different ranges. The reported results are averaged over 100 random seeds. It can be seen that a larger random sample range improves the chance of finding global optima. Similar phenomenon also exist in practice as shown in Appendix A.1.

the perturbation on the action space and produces diverse strategies (*a population*), and we choose those well-performed strategies to distill their knowledge into π_T (a *selection*). Fig.**??** provides an illustration of the learning pipeline and Algorithm 2 presents the detailed learning procedure of ESPD.

It is worth noting that the term *Evolutionary* in ESPD is used to demonstrate the **selection** procedure. Moreover, although plenty of policy gradient methods leverage action space noise in exploration [45, 65, 26], there is a clear gap between those perturbation and the evolutionary insight in our methods: The most important difference between action space noise in ESPD and previous policy gradient methods is that ESPD regards those **stochastic variants** as from a series of unknown **oracle deterministic policies**, then use a selection function as a filter the well-performed unknown oracle policies and finally distill those transitions to the present policy. On the other hand, previous policy gradient methods do not contain such a selection step, but instead use **all** those variants to estimate the value function and improve the policy through policy gradient accordingly.

3.3 Zeroth-Order Self-Supervised Continuous Control

3.3.1 A Motivating Example

Figure 3.4 shows a motivating example to demonstrate the benefits of applying zeroth-order optimization to policy updates. Consider we have learned a Q function that has multiple local optima[1], and our present deterministic policy selects a certain action at this state, denoted as the red dot in Figure 3.4(a). In deterministic policy gradient methods [68, 45], the policy gradient is conducted according to the chain rule to update the policy parameter θ with regard to Q-value by timing up the Jacobian matrix $\nabla_\theta \pi_\theta(s)$ and the derivative of Q, *i.e.*, $\nabla_a Q(s, a)$. Consequently, the policy gradient can only lead to a local improvement, and similar local improvement behaviors are also observed in stochastic policy gradient methods like PPO and SAC [65, 31, 75, 18]. Instead, if we can sample suf-

[1]Here we assume the traditional estimation of Q function is sufficient for a global exploitation [26, 31] and we will discuss an improved estimation method in the next section.

Algorithm 3 Zeroth-Order Policy Optimization

```
**Require**
Objective function $Q_s$, domain $\mathcal{A}$, current point $a_0 = \pi_\theta(s)$, number of local samples $n_1$ and global samples $n_2$, local scale $\eta > 0$ and step size $h$.
**Locally sampling**
Sample $n$ points around $a_0$ by
    $a_i = a_0 + \eta e_i$ for $e_i \sim \mathcal{N}(a_0, I_d), i = 1, \ldots n_1,$
where $\mathcal{N}(a_0, I_d)$ is the standard normal distribution centered at $a_0$.
**Globally sampling**
Sample $n$ points uniformly in the entire space by
    $a_{i+n_1} \sim \mathcal{U}_\mathcal{A},$ for $i = 1, \ldots, n_2,$
where $\mathcal{U}_\mathcal{A}$ is the uniform distribution over $\mathcal{A}$.
**Update**
Set $a^+ =_{a \in \{a_0, \ldots a_{n_1+n_2}\}} Q_s(a)$.
Update policy $\pi_\theta$ according to Eq.(3.18)
```

ficient *random* actions in a broader range of the action space, denoted as blue dots in Figure 3.4(b), and then evaluate their values respectively through the learned Q estimator, it is possible to find the action with a higher Q value as the target action in the optimization. Figure 3.4(c) shows the simulation result using different sample ranges for the sample-based optimization starting from the red point. It is clear that a larger sample range improves the chance of finding the global optima. Utilizing such a global exploitation on the learned value function is the key insight of this work.

3.3.2 Zeroth-Order Supervised Policy Improvement

Q-learning in the tabular setting relies on finding the best action given the current state, which can be difficult in the continuous action space due to the non-convexity of Q. Instead, a policy network is thus trained to approximate the solution. In most of previous policy gradient methods, the policy class is selected to be Gaussian in consideration of both exploration and computational tractability, while, in this work, we consider the deterministic policy class which is simpler and easier to learn as presented in [68].

As shown in [68], DPG updates the policy network only through the first-order gradient of the current Q value estimation:

$$\nabla_a Q_{w_t}(s_t, \pi_{\theta_t}(s_t)) \nabla_\theta \pi_{\theta_t}(s_t). \tag{3.17}$$

Such an update through the local information of Q may incur a slow convergence rate, especially when Q function is non-convex. To mitigate this issue, we propose the Zeroth-Order Supervised Policy Improvement (ZOSPI), which exploits the entire learned Q function instead of merely the local information of Q. Thus the key insight of our proposed method is to utilize the zeroth-order method to overcome the local policy improvement problem induced by the non-convexity of Q.

To be specific, we first calculate the predicted action $a_0 = \pi_{\theta_t}(s_t)$. Then we sample two sets of actions with size n, namely a local set and a global set. For the local set, we sample actions randomly from a Gaussian distribution centered at a_0. For the global set, we sample points uniformly over the action space. The update a_t^+ is chosen as the action that gives the highest Q value in the union of two sets. Finally, we apply the supervised policy improvement that minimizes the L_2 distance between a_t^+ and $\pi_{\theta_t}(s_t)$, which gives the descent direction:

$$\nabla_\theta \frac{1}{2}(a_t^+ - \pi_{\theta_t}(s_t))^2 = (a_t^+ - \pi_{\theta_t}(s_t))\nabla_\theta \pi_{\theta_t}(s_t). \tag{3.18}$$

The implementation detail is shown in Algorithm 3.

3.3.3 Comparison between two methods

We now compare the performances obtained via applying Eq.(3.18) to the standard deterministic policy gradient update used in Eq.(3.17). Following the definition in Algorithm 3, let the best action in the local set be $a_L =_{a_i, i=1,\dots n} Q_{w_t}(s_t, a_i)$. Assume $Q_{w_t}(s_t, \cdot)$ is continuous for any s_t and w_t. We have $\lim_{n\to\infty} \lim_{\eta\to 0}(a_L - a_0)/\eta \propto \nabla_a Q_{w_t}(s_t, a_0)$.

Proposition 3.3.3, whose proof is provided in Appendix A.3, guarantees that with a sufficiently large number of samples and a sufficiently small η, zeroth-order optimization can at least find the local descent direction as in a first-order method.

When the best action is actually included in the global set, zeroth-order optimization will update towards the global direction with a larger step size as $\mathbb{E}\|a_1 - \pi_{\theta_t}(s_t)\| \leq \mathbb{E}\|a_{n+1} - \pi_{\theta_t}(s_t)\|$ in general. *The benefits of ZOSPI over DPG are determined by the probability of sampling an action globally that is close to the global minima.*

With a sufficient number of sampled actions at each step, ZOSPI is able to find better solutions in terms of higher Q values for a given state, and therefore can globally exploit the Q function, which is especially useful when the Q function is non-convex as illustrated in Figure 3.4.

Algorithm 4 provides the pseudo code for ZOSPI, where we follow the double Q network in TD3 as the critic, and also use target networks for stability. In the algorithm we sample actions in the global set from a uniform distribution on the action space $a_i \sim \mathcal{U}_\mathcal{A}$, and sample actions from an on-policy local Gaussian (e.g., $a_i(s) = \pi_{\theta_{old}}(s) + \eta_i$, and $\eta_i \sim \mathcal{N}(0, \sigma^2)$)) to form the local set and guarantee local exploitation, so that ZOSPI will at least perform as good as deterministic policy gradient methods [68, 45, 26].

3.3.4 Discussion on the Benefits of Global Sampling

In this section, we discuss a type of structure, with which our zeroth-order optimization has a exponential convergence rate. To better explain our points, we include an Algorithm 5 in Appendix A.2, a modified version of Algorithm 3, which prevents the sampled action jumping too far across different global regions.

[Sampling-Easy Functions] A function $F : \mathcal{X} \subset \mathbb{R}^d \mapsto \mathbb{R}$ is called $\alpha\beta$-*Sampling-Easy*, if it has an unique global minima x^*, and there exists an region $\mathcal{D} \subset \mathcal{X}$, such that

1. $x^* \in \mathcal{D}$;
2. F is α-convex and β-smooth in region $\mathcal{D}$;
3. $|\mathcal{D}|/|\mathcal{X}| \geq c/d$ for some $c > 0$.

A function F is α-convex in region $\mathcal{D}$, if $F(y) \geq F(x) + \langle \nabla F(x), y - x\rangle + \frac{\alpha}{2}\|y - x\|^2, x, y \in \mathcal{D}$. Furthermore, it is β-smooth, if $|F(y) - F(x) - \langle \nabla F(x), y - x\rangle| \leq \frac{\beta}{2}\|x - y\|^2, x, y \in \mathcal{D}$.

Algorithm 4 Zeroth-Order Supervised Policy Improvement (ZOSPI)

```
1: Require
2:   Number of epochs $M$, size of mini-batch $N$, momentum $\tau > 0$.
3:   Random initialized policy network $\pi_\theta$, target policy network $\pi_{\theta'}$, $\theta' \leftarrow \theta$.
4:   Two random initialized $Q$ networks, and corresponding target networks, parameterized by $w_1, w_2, w'_1, w'_2$. $w'_i \leftarrow w_i$.
5:   Empty experience replay buffer $\mathcal{D} = \{\}$.
6: for iteration $= 1, 2, ...$ do
7:   for t $= 1, 2, ..., T$ do
8:     # Interaction
9:     Run policy $\pi_\theta$ in environment, store transition tuples $(s_t, a_t, s_{t+1}, r_t)$ into $\mathcal{D}$.
10:    for epoch $= 1, 2, ..., M$ do
11:      Sample a mini-batch of transition tuples $\mathcal{D}' = \{(s_{t_j}, a_{t_j}, s_{t_j+1}, r_{t_j})\}_{j=1}^N$.
12:      # Update $Q$
13:      Calculate target $Q$ value $y_j = r_{t_j} + \min_{i=1,2} Q_{w'_i}(s_{t_j+1}, \pi_{\theta'}(s_{t_j}))$.
14:      Update $w_i$ with one step gradient descent on the loss $\sum_j (y_j - Q_{w'_{i,}}(s_{t_j}, a_{t_j}))^2$, $i = 1, 2$.
15:      # Update $\pi$
16:      Call Algorithm 3 for policy optimization to update $\theta$.
17:    end for
18:    $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$.
19:    $w'_i \leftarrow \tau w_i + (1-\tau) w'_i$.
20:    $w_i \leftarrow w_i$; $\theta \leftarrow \theta$.
21:  end for
22: end for
```

For any $\alpha\beta$-Sampling-Easy function F that satisfies $F(x^*) \le F(x) - \epsilon_0$, for all $x \notin \mathcal{D}$, by running Algorithm 5, on average it requires at most

$$O\left(\log\left(\frac{D_m \beta}{\min\{\epsilon, \epsilon_0\}}\right)\frac{d^2\beta}{c\alpha}\right)$$

iterations to find an ϵ-optimal solution for any $\epsilon > 0$, with c and d the same in Definition 3.3.4. Here $D_m = \max_{x \in \mathcal{D}} \|x - x^*\|_2^2$. The proof is provided in Appendix A.2.

Theorem 3.3.4 suggests that despite the function is non-convex or non-smooth, the convergence can be guaranteed as long as there is a sufficiently large convex and smooth area around the global optima.

3.4 Benefits of Continuous Control without Policy Gradient

Different from standard policy gradient methods, the policy optimization step in ZOSPI can be interpreted as sampling-based supervised learning. Such difference provides several potential benefits and enable further extensions of this work. Here we discuss the combination of ZOSPI with UCB exploration as well as non-parametric models in RL.

3.4.1 Better Exploration with Bootstrapped Networks

Sample efficient RL requires algorithms to balance exploration and exploitation. One of the most popular way to achieve this is called optimism in face of uncertainty (OFU) [14, 36, 8, 37], which gives an upper bound on Q estimates and applies the optimal action corresponding to the upper bound. The optimal action a_t is given by the following optimization problem:

$$ {}_a Q^+(s_t, a), \tag{3.19}$$

where Q^+ is the upper confidence bound on the optimal Q function. A guaranteed exploration performance requires both a good solution for (3.19) and a valid upper confidence bound.

While it is trivial to solve (3.19) in the tabular setting, the problem can be intractable in a continuous action space. Therefore, as shown in the previous section, ZOSPI adopts a local set to approximate policy gradient descent methods in the local region and further applies a global sampling scheme to increase the potential chance of finding a better maxima.

As for the requirement of a valid upper confidence bound, we use bootstrapped Q networks to address the uncertainty of Q estimates as in [52, 51, 2, 40, 18]. Specifically, we keep K estimates of Q, namely $Q_1, \dots Q_K$ with bootstrapped samples from the replay buffer. Let $\overline{Q} = \frac{1}{K}\sum_k Q_k(s,a)$. An upper bound Q^+ is

$$Q^+(s,a) = \overline{Q} + \phi\sqrt{\frac{1}{K}\sum_k [Q_k(s,a) - \overline{Q}]^2}, \tag{3.20}$$

where ϕ is the hyper-parameter controlling the failure rate of the upper bound. Another issue is on the update of bootstrapped Q networks. Previous methods [2] usually update each Q network with the following target $r_t + \gamma Q_k\left(s_{t+1}, \pi_{\theta_t}(s_{t+1})\right)$, which violates the Bellman equation as π_{θ_t} is designed to be the optimal policy for Q^+ rather than Q_k. Using π_{θ_t} also introduces extra dependencies among the K estimates. We instead employ a global random sampling method to correct the violation as

$$r_t + \gamma \max_{i=1,\dots n} Q_k\left(s_{t+1}, a_i\right), \quad a_1, \dots a_n \sim \mathcal{U}_{\mathcal{A}}.$$

The correction also reinforces the argument that a global random sampling method yields a good approximation to the solution of the optimization problem (3.19). The detailed algorithm is provided in Algorithm 6 in Appendix A.6.

3.4.2 Gaussian Processes for Continuous Control

Different from previous policy gradient methods, the self-supervised learning paradigm of ZOSPI permits it to learn both its actor and critic with a regression formulation. Such a property enables the learning of actor in ZOSPI to be implemented with either parametric models like neural networks or non-parametric models like Gaussian Processes (GP). Although plenty of previous works have discussed the application of GP in RL by virtue of its natural uncertainty capture ability, most of these works are limited to model-based methods or discrete action spaces for value estimation [42, 24, 41, 44, 28, 25]. On the other hand, ZOSPI formulates the policy optimization in continuous control tasks as a regression objective, therefore empowers the usage of GP policy in continuous control tasks.

As a first attempt of applying GP policies in continuous control tasks, we simply alter the actor network with a GP to interact with the environment and collect data, while the value approximator is still parameterized by a neural network. We leave the investigation of better consolidation design in future work.

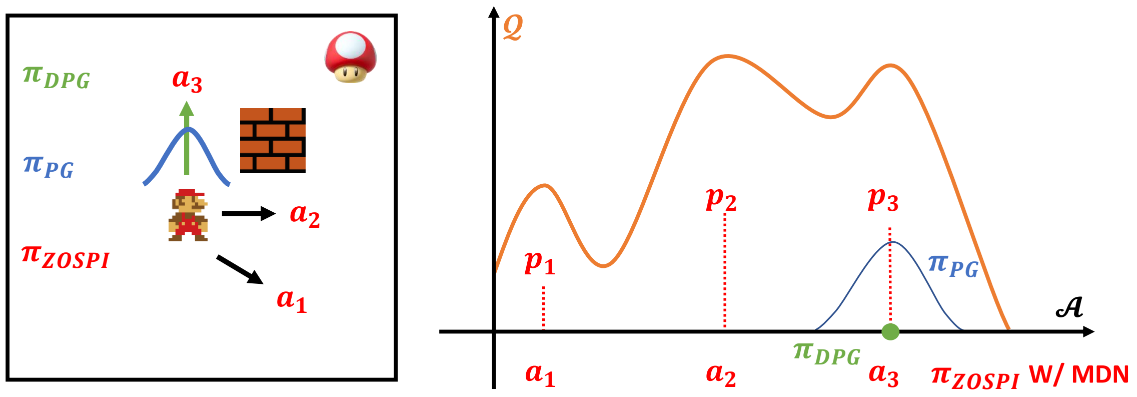


Figure 3.5: Illustration of how ZOSPI with MDN work when multiple optimal actions exist for a certain state: (left) Supermario is going to collect the mushroom, both a_2 and a_3 are optimal action for the current location. While deterministic policy gradient methods (green color) are only able to learn one of those optimal actions, ZOSPI with MDN is able to learn both. Stochastic policy gradient methods normally learn a Gaussian policy class.

3.4.3 Learning Multi-Modal Policies with Mixture Density Networks

In the context of RL, there might be multiple optimal actions given a certain state, e.g., stepping upward and then right-ward may lead to the same state as stepping right and then upward, therefore both choices result in the same return.

The Mixture Density Networks (MDNs) is proposed by [12] for multi-modal regression. Applying MDNs in ZOSPI can lead to multi-modal policies: different from normal Dirac policy parameterization used in TD3, ZOSPI with MDN predicts a mixture of Dirac policies, i.e., the policy $\pi_{MDN}(s)$ predicts K choices for action $\vec{a} = \{a_1, ..., a_K\} \in \mathcal{A}^K$, with their corresponding probability $\vec{p} = \{p_1, ..., p_K\} \in \mathbb{R}^K$, and $\sum_i^K p_i = 1$. And the action is then sampled by

$$\pi_{MDN}(s) = a_i, \quad \text{w.p.} \;\; p_i \tag{3.21}$$

Thereby, instead of learning the mean value of multiple optimal actions, ZOSPI with MDN is able to learn multiple optimal actions. And different from previous stochastic multi-modal policies discussed in [31], our learning of multi-modal policy do not aim to fit the entire Q function distribution. Rather, ZOSPI with MDN focus on learning the multi-modality in best choices of actions, thus circumvent the computational difficulty in generating multi-modal policies [30, 75]. Figure 3.5 illustrates how ZOSPI with MDN distinguishes from previous policy gradient methods by learning multi-modal policies.

Chapter 4

Results

4.1 Experiments of PCHID

As a policy $\pi(s, g)$ aims at reaching a state s' where $m(s') = g$, by intuition the difficulty of solving such a goal-oriented task depends on the complexity of m. In Sec.4.1.1 we start with a simple case where m is an identical mapping in the environment of GridWorld by showing the agent a fully observable map. Moreover, the GridWorld environment permits us to use prior knowledge to calculate the accuracy of any TEST function. We show that PCHID can work independently or augmented with the DQN in discrete action space setting, outperforming the DQN as well as the DQN augmented with HER. The GridWorld environment corresponds to the identical mapping case $\mathcal{G} = \mathcal{S}$. In Sec.4.1.2 we test our method on a continuous control problem, the FetchReach environment provided by Plappert et al. [57]. Our method outperforms PPO by achieving 100% successful rate in about 100 episodes. We further compare the sensitivity of PPO to reward values and the robustness PCHID owns. The state-goal mapping of FetchReach environment is $\mathcal{G} \subset \mathcal{S}$.

4.1.1 GridWorld Navigation

We use the GridWorld navigation task in Value Iteration Networks (VIN) [73], in which the state information includes the position of the agent, and an image of the map of obstacles and goal position. In our experiments we use 16×16 domains, navigation in which is not an effortless task. Fig.3.1(c) shows an example of our domains. The action space is discrete and contains 8 actions leading the agent to its 8 neighbour positions respectively. A reward of 10 will be provided if the agent reaches the goal within 50 timesteps, otherwise the agent will receive a reward of -0.02. An action leading the agent to an obstacle will not be executed, thus the agent will stay where it is. In each episode, a new map will randomly selected start s and goal g points will be generated. We train our agent for 500 episodes in total so that the agent needs to learn to navigate within just 500 trials, which is much less than the number used in VIN [73].[1] Thus we can demonstrate the high data efficiency of PCHID by testing the learned agent on 1000 unseen maps. Our work follows VIN to use the rollout success rate as the evaluation metric.

[1]Tarmar et al. train VIN through the imitation learning (IL) with ground-truth shortest paths between start and goal positions. Although both of our approaches are based on IL, we do not need ground-truth data

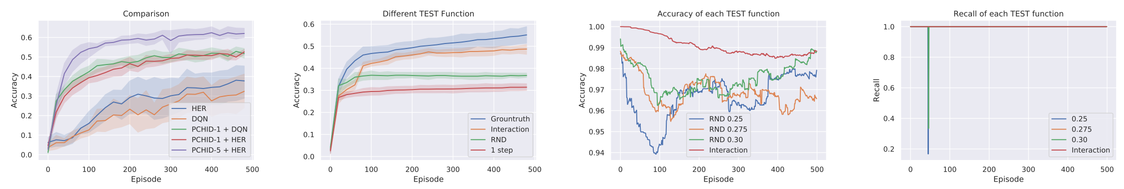


Figure 4.1: (a): The rollout success rate on test maps in 10 experiments with different random seeds. HER outperforms VIN, but the difference disappears when combined with PCHID. PCHID-1 and PCHID-5 represent 1-step and 5-step PCHID. (b): Performance of PCHID module alone with different TEST functions. The blue line is from ground truth testing results, the orange line and green line are Interaction and RND respectively, and the red line is the 1-step result as a baseline. (c)(d): Test accuracy and recall with Interaction and RND method under different threshold.

Our empirical results are shown in Fig.4.1. Our method is compared with DQN, both of which are equipped with VIN as policy networks. We also apply HER to DQN but result in a little improvement. PC with 1-step HID, denoted by PCHID 1, achieves similar accuracy as DQN in much less episodes, and combining PC with 5-step HID, denoted by PCHID 5, and HER results in much more distinctive improvement.

4.1.2 OpenAI Fetch Env

In the Fetch environments, there are several tasks based on a 7-DoF Fetch robotics arm with a two-fingered parallel gripper. There are four tasks: FetchReach, FetchPush, FetchSlide and Fetch-PickAndPlace. In those tasks, the states include the Cartesian positions, linear velocity of the gripper, and position information as well as velocity information of an object if presented. The goal is presented as a 3-dimentional vector describing the target location of the object to be moved to. The agent will get a reward of 0 if the object is at the target location within a tolerance or -1 otherwise. Action is a continuous 4-dimentional vector with the first three of them controlling movement of the gripper and the last one controlling opening and closing of the gripper.

FetchReach Here we demonstrate PCHID in the FetchReach task. We compare PCHID with PPO and HER based on PPO. Our work is the first to extend hindsight knowledge into on-policy algorithms [57]. Fig.4.2 shows our results. PCHID greatly improves the learning efficiency of PPO. Although HER is not designed for on-policy algorithms, our combination of PCHID and PPO-based HER results in the best performance.

4.1.3 Combing PCHID with Other RL Algorithms

As PCHID only requires sufficient exploration in the environment to approximate optimal sub-policies progressively, it can be easily plugged into other RL algorithms, including both on-policy algorithms and off-policy algorithms. At this point, the PCHID module can be regarded as an extension of HER for off-policy algorithms. We put forward three combination strategies and evaluate each of them on both GridWorld and FetchReach environment.

Joint Training The first strategy for combining PCHID with normal RL algorithm is to adopt a shared policy between them. A shared network is trained through both temporal difference learning

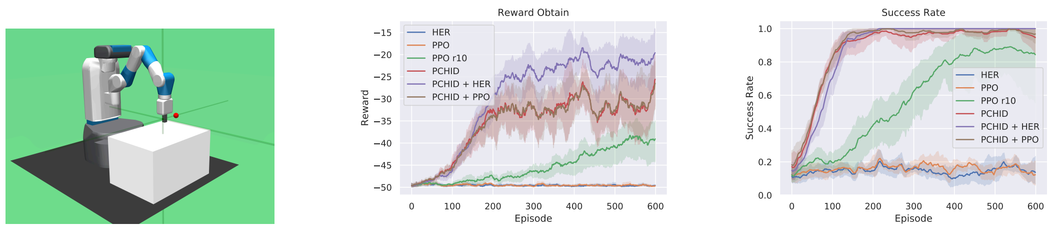


Figure 4.2: (a): The FetchReach environment. (b): The reward obtaining process of each method. In PPO r10 the reward of achieving the goal becomes 10 instead of 0 as default, and the reward is re-scaled to be comparable with other approaches. This is to show the sensitivity of PPO to reward value. By contrast, the performance of PCHID is unrelated to reward value. (c): The success rate of each method. Combining PPO with PCHID brings about little improvement over PCHID, but combining HER with PCHID improves the performance significantly.

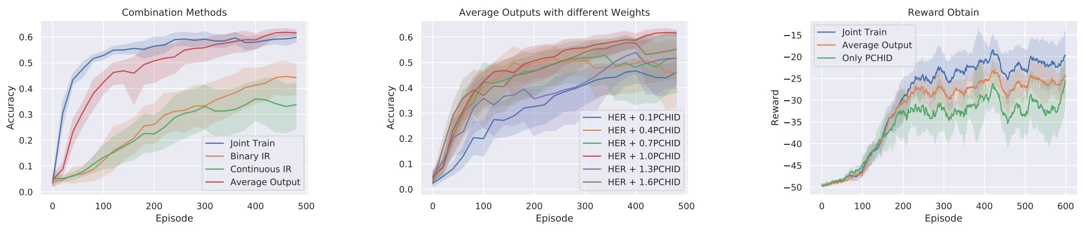


Figure 4.3: (a): Accuracy of GridWorld under different combination strategies. (b): Averaging outputs with different weights. (c): Obtained Reward of FetchReach under different strategies.

in RL and self-supervised learning in PCHID. The PCHID module in joint training can be viewed as a regularizer.

Averaging Outputs Another strategy for combination is to train two policy networks separately, with data collected in the same set of trajectories. When the action space is discrete, we can simply average the two output vectors of policy networks, e.g. the Q-value vector and the log-probability vector of PCHID. When the action space is continuous, we can then average the two predicted action vectors and perform an interpolated action. From this perspective, the RL agent here actually learns how to work based on PCHID and it parallels the key insight of ResNet [32]. If PCHID itself can solve the task perfectly, the RL agent only needs to follow the advice of PCHID. Otherwise, when it comes to complex tasks, PCHID will provide basic proposals of each decision to be made. The RL agent receives hints from those proposals thus the learning becomes easier.

Intrinsic Reward (IR) This approach is quite similar to the curiosity driven methods. Instead of using the inverse dynamics to define the curiosity, we use the prediction difference between PCHID module and RL agent as an intrinsic reward to motivate RL agent to act as PCHID. Maximizing the intrinsic reward helps the RL agent to avoid aimless explorations hence can speed up the learning process.

Fig.4.3 shows our results in GridWorld and FetchReach with different combination strategies. Joint training performs the best and it does not need hyper-parameter tuning. On the contrary, the averaging outputs requires determining the weights while the intrinsic reward requires adjusting its scale with regard to the external reward.

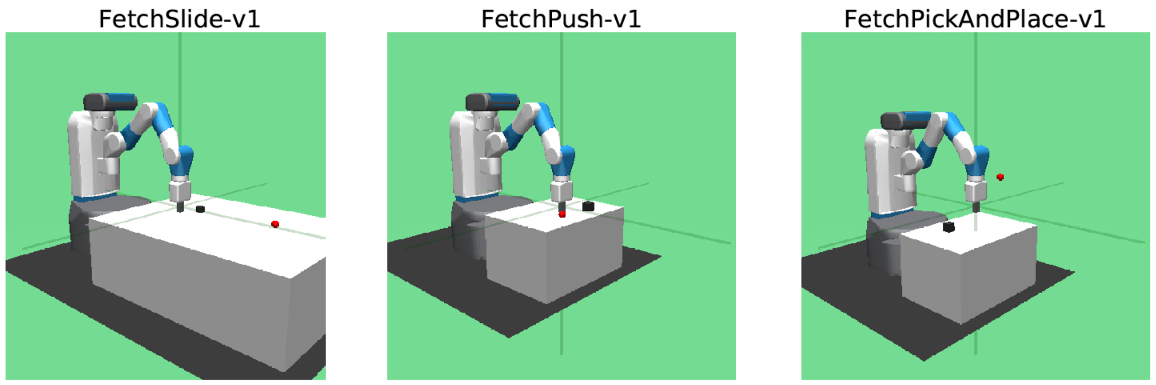


Figure 4.4: Three robotic manipulation environments. FetchPush-v1: push a block to a goal position, FetchSlide-v1: slide a puch to a goal position and FetchPickAndPlace-v1: lift a block into the air.

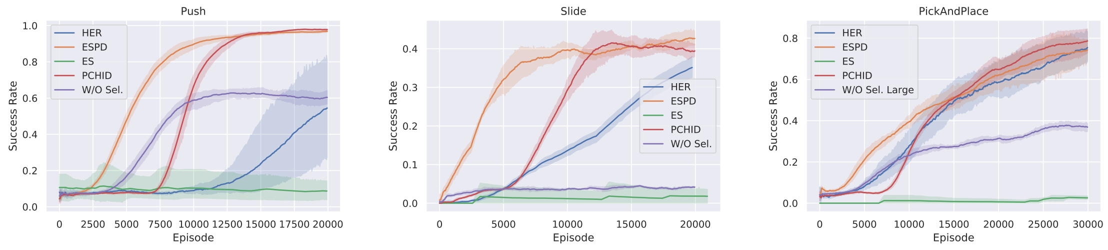


Figure 4.5: The test success rate comparison on the FetchPush-v1, FetchSlide-v1 and FetchPickAndPlace-v1 among our proposed method (ESPD), ESPD without the SELECT function (W/O Sel.) PCHID, HER and Evolution Strategy (ES).

4.2 Experiments of ESPD

4.2.1 Result on the Fetch Benchmarks

We demonstrate the proposed method on the Fetch Benchmarks. Specifically, we evaluate our method on the FetchPush, FetchSlide and FetchPickAndPlace environments, as shown in Fig.4.4. We compare our proposed method with the HER [4, 55] released in OpenAI Baselines [22] and the Evolution Strategy [60] which is a counterpart of our method with parameter noise. We also include PCHID as a baseline, however as PCHID [70] can be regarded as a special case of ESPD if we gradually increase the hyper-parameter Horizon in ESPD from 1 to K, the performance of PCHID is upper-bounded by ESPD. Such result can also be inferred from our ablation study on the Horizon K in the next section, which shows smaller K limits the performance, and achieves worse learning efficiency than $K = 8$, the default hyper-parameter used in ESPD.

Fig.4.5 shows the comparison of different approaches. For each environment, we conduct 5 experiments with different random seeds and plot the averaged learning curve. In Fig.4.5, we also show the ablated experiment result by turning off the SELECT step in ESPD, where the result is denoted as W/O Sel. Such a ablation study demonstrates the importance of the SELECT step. ESPD shows superior learning efficiency and can learn to solve the task in fewer episodes in all the three environments.

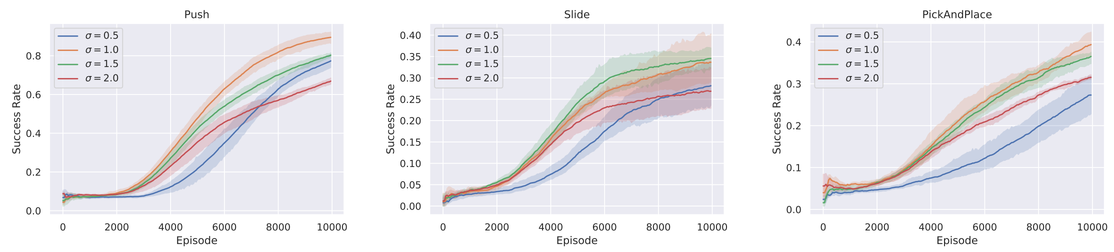


Figure 4.6: The test success rate comparison on the FetchPush-v1, FetchSlide-v1 and FetchPickAndPlace-v1 with different scale of exploration factors.

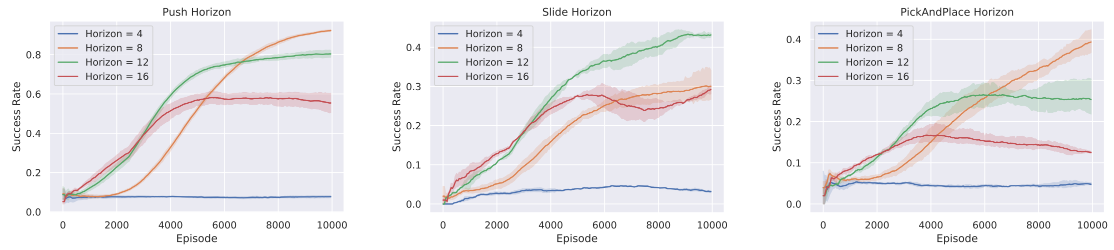


Figure 4.7: The test success rate comparison on the FetchPush-v1, FetchSlide-v1 and FetchPickAndPlace-v1 with different scales of Horizon K.

4.2.2 Sensitivity to Hyper-Parameter

Exploration Factor The exploration factor σ controls the randomness of behavior policy and therefore determines the behavior of generated samples. While larger σ helps the agents to benefit exploration by generating samples with large variance, smaller σ helps to generate a biased sample with little variance. Here we need to select a proper σ to balance the variance and bias. Fig.4.6 shows our ablation study on the selection of different exploration factors. The results are generated with 5 different random seeds. We find in all environments, the exploration factor $\sigma = 1$ provides sufficient exploration and relatively high learning efficiency.

Horizon K In our proposed method, the parameter of Horizon K determines the maximal length of sample trajectories the policy can learn from. Intuitively, smaller K decreases the learning efficiency as the policy is limited by its small horizon, making it hard to plan for the tasks that need more steps to solve. On the other hand, larger K will provide a better concept of the local as well as global geometry of the state space, and thus the agent may learn to solve more challenging tasks. However, using large K introduces more interactions with the environment, and needs more computation time. Moreover, as the tasks normally do not need lots of steps to finish, when the Horizon is getting too large, more noisy actions will be collected and be considered as better solutions and hence impede the learning performance. Fig.4.7 shows our ablation studies on the selection of Horizon K. The results are generated with 5 different random seeds. We find that $K = 8$ provides satisfying results in all of the three environments.

It is worth noting that HID can be seen as a special case of ESPD if we gradually increase the hyper-parameter Horizon in ESPD from 1 to K. From this perspective, as shown in Fig.4.7 where at the beginning of learning a smaller Horizon leads to bad performance, the performance of HID is upper bounded by ESPD.

4.3 Experiments for ZOSPI

In this section, we conduct experiments on a diagnostic 2-D navigation environment named Four-Solution-Maze and five MuJoCo locomotion benchmarks to demonstrate the effectiveness as well as the flexibility of the proposed method. Specifically, we validate the following statements:

1. If we use ZOSPI with locally sampled actions, the performance of ZOSPI should be the same as its policy gradient counterpart (i.e., TD3); if we increase the sampling range, ZOSPI will be able to better exploit the Q function and find better solutions than the methods based on policy gradient.

2. If we continuously increase the sampling range, it will result in an uniform sampling (in practice we include an additional local sampling, a coarse to fine paradigm to encourage local improvements in the later stage of the learning process), and the Q function can be maximally exploited.

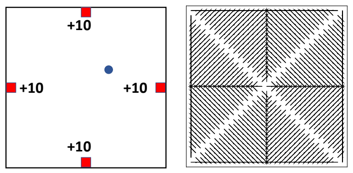
3. ZOSPI can be combined with Bootstrapped Q-value estimation and behave with the principle OFU [14] to pursue better exploration, or combined with GP and lay a foundation of future works of continuous control with non-parametric models, or work with MDNs to learn multi-modal policies.

4.3.1 Experiments on the Four-Solution-Maze.

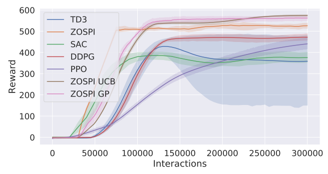
The Four-Solution-Maze (FSM) environment is a diagnostic environment where four positive reward regions with a unit side length are placed in the middle points of 4 edges of a $N \times N$ map. An agent starts from a uniformly initialized position in the map and can then move in the map by taking actions according to the location observations (current coordinates x and y). Valid actions are limited to $[-1, 1]$ for both x and y axes. Each game consists of $2N$ timesteps for the agent to navigate in the map and collect rewards. In each timestep, the agent will receive a $+10$ reward if it is inside one of the 4 reward regions or a tiny penalty otherwise. For simplicity, there are no obstacles in the map, the optimal policy thus will find the nearest reward region, directly move towards it, and stay in the region till the end. Figure 4.8(a) visualizes the environment and the ground-truth optimal solution.

Although the environment is simple, we found it extremely challenging due to existence of multiple sub-optimal policies that only find some but not all four reward regions. We do not conduct grid search on hyper-parameters of the algorithms compared in our experiments but set them to default setting across all experiments. Though elaborated hyper-parameter tuning may benefit for certain environment.

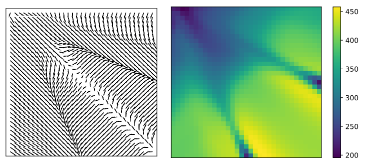
On this environment we compare ZOSPI to on-policy and off-policy SOTA policy gradient methods in terms of the learning curves, each of which is averaged by 5 runs. The results are presented in Figure 4.8(b). And learned policies from different methods are visualized in Figure 4.8(c)-4.8(i). For each method we plot the predicted behaviors of its learned policy at grid points using arrows (although the environment is continuous in the state space), and show the corresponding value function of its learned policy with a colored map. All policies and value functions are learned with 0.3M interactions except for SAC whose figs are learned with 1.2M interactions as it can find 3 out of 4 target regions when more interactions are provided.



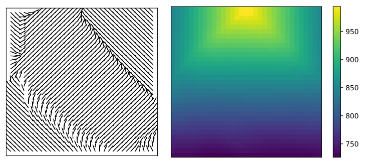
(a) Env. and the optimal policy



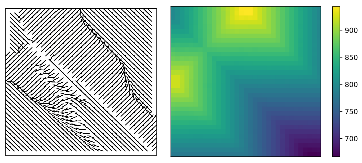
(b) Performance comparison



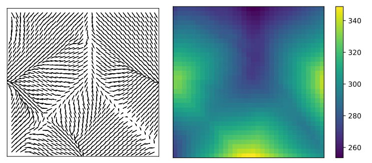
(c) PPO



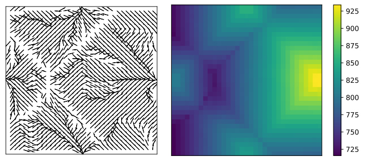
(d) DDPG



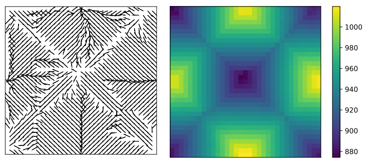
(e) TD3



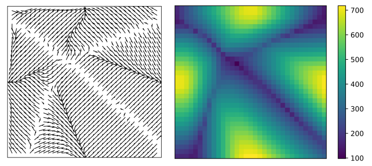
(f) SAC



(g) ZOSPI



(h) ZOSPI-UCB



(i) ZOSPI-GP

Figure 4.8: Visualization of learned policies on the FSM environment. (a) the FSM environment and its optimal solution, where the policy should find the nearest reward region and move toward it; (b) learning curves of different approaches; (c)-(i) visualize the learned policies and corresponding value functions. We run multiple repeat experiments and show the most representative and well-performing learned value function and policy of each method.

We use 4 bootstrapped Q networks for the upper bound estimation in consideration of both better value estimation and computational cost for ZOSPI with UCB. And in ZOSPI with GP, a GP model is used to replace the actor network in data-collection, i.e.,exploration. The sample efficiency of ZOSPI is much higher than that of other methods. Noticeably ZOSPI with UCB exploration is the only method that can find the optimal solution, *i.e.*, a policy directs to the nearest region with a positive reward. All other methods get trapped in sub-optimal solutions by moving to only part of reward regions they find instead of moving toward the nearest one.

4.3.2 ZOSPI on the MuJoCo Locomotion Tasks.

In this section we evaluate ZOSPI on the OpenAI Gym locomotion tasks based on the MuJoCo engine 1606.01540,TodorovET12. Concretely, we test ZOSPI on five locomotion environments, namely the Hopper-v2, Walker2d-v2, HalfCheetah-v2, Ant-v2 and Humanoid-v2 and include TD3 and SAC, respectively the deterministic and the stochastic SOTA policy gradient methods in the comparison. We also include PPO and OAC [18] to better expose the learning efficiency of ZOSPI. We compare results of different methods within 0.5M environment interactions to demonstrate the high learning efficiency of ZOSPI except in the Humanoid-v2 we run 1M interactions. The results of TD3 and OAC are obtained by running the code released by the authors, results of PPO is gained through the high quality open-source implementation of Baselines [22], and the results of SAC are extracted from the training logs of haarnoja2018soft.

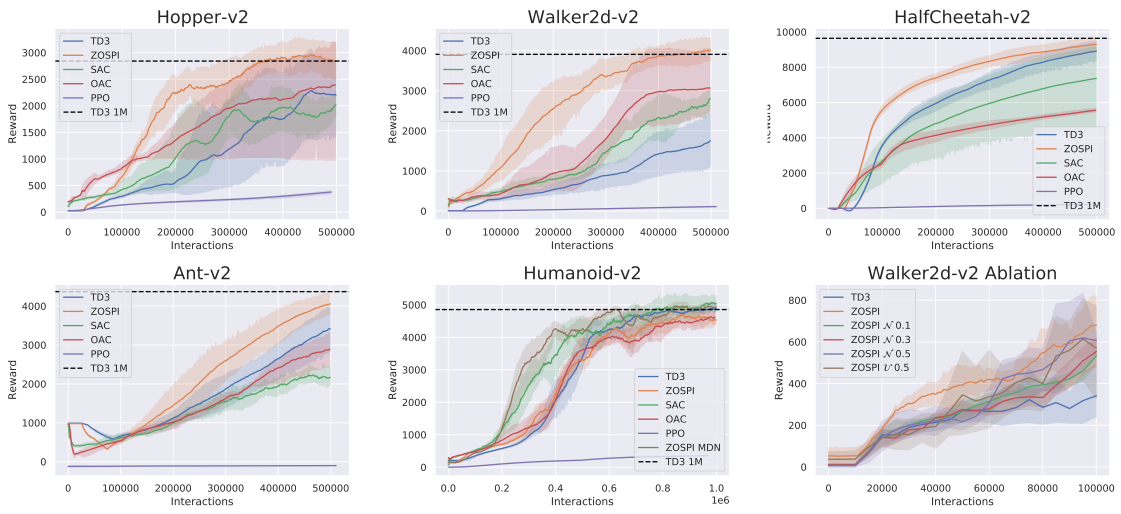


Figure 4.9: Experimental results on the MuJoCo locomotion tasks. The shaded region represents half a standard deviation. The dashed lines indicate asymptotic performance of TD3 after 1M interactions. ZOSPI is able to reach on-par performance within much less interactions.

The results are included in Figure 4.9. It is worth noting that in our implementation of ZOSPI, only 50 actions are sampled for all tasks and it is sufficient to learn well-performing policies. Surprisingly, with such high sampling efficiencies, the results of ZOSPI are good even in the tasks that have high-dimensional action spaces such as Ant-v2 and Humanoid-v2. While a total of 50 sampled actions should be very sparse in the high dimensional space, we contribute the success of ZOSPI to the generality of the policy network as well as the sparsity of meaningful actions, *i.e.*, even in tasks that have high dimension action spaces, only limited dimensions of actions are crucial for making decisions.

In the environment of Humanoid-v2, ZOSPI with MDN is used to further improve the learning efficiency as discussed in Section 3.4.3. Although the vanilla ZOSPI under-performs SAC, ZOSPI with MDN achieves remarkably improved sample efficiency. Implementation details of ZOSPI with MDN can be found in Appendix A.5

The last plot in Figure 4.9 shows the ablation study on the sampling range $\mathcal{N}$ in ZOSPI, where a sampling method based on a zero-mean Gaussian is applied and we gradually increase its variance from 0.1 to 0.5. We also evaluate the uniform sampling method with radius of 0.5, which is denoted as $\mathcal{U}$ 0.5 in the Figure. The results suggest that zeroth-order optimization with local sampling performs similarly to the policy gradient method, and increasing the sampling range can effectively improve the performance.

Chapter 5

Conclusion

In the first part of this work we propose the Policy Continuation with Hindsight Inverse Dynamics (PCHID) to solve the goal-oriented reward sparse tasks from a new perspective. Our experiments show the PCHID is able to improve data efficiency remarkably in both discrete and continuous control tasks. Moreover, our method can be incorporated with both on-policy and off-policy RL algorithms flexibly.

The second part of this work developed a practical algorithm that can evolve to solve the GCRS problems by distilling knowledge from a series of the stochastic variants of a learned agent. The key insight behind our proposed method is based on the SDE formulation of the GCRS tasks: such tasks can be solved by learning to reduce the First Hitting Time. Based on the formulation, we proposed the method of Evolutionary Stochastic Policy Distillation (ESPD), which is composed of an evolutionary action space perturbation, a select procedure as a filter to collect data, and a policy distillation learning step to optimize the policy. Our experiments on the OpenAI Fetch benchmarks show that the proposed method has a much improved sample efficiency as well as stability with regard to previous baseline methods, namely the PCHID, the Evolution Strategies, and Hindsight Experience Replay.

In the last part, we propose the method of Zeroth-Order Supervised Policy Improvement (ZOSPI) as an alternative approach of policy gradient algorithms for continuous control. ZOSPI improves the learning efficiency of previous policy gradient learning by exploiting the learned Q functions globally and locally through sampling in the action space. Different from previous policy gradient methods, the policy optimization of ZOSPI is based on supervised learning so that the learning of actor can be conducted with regression. Such a property enables several extensions such like optimistic exploration, non-parametric policies and multi-modal policies to be seamlessly cooperated with ZOSPI, and thereby opens up potential future directions. We evaluate ZOSPI on a diagnostic environment called Four-Solution-Mazethe and five MuJoCo locomotion tasks, where our method achieves improved performance in terms of sample efficiency and asymptotic performance compared to SOTA policy gradient methods.

Appendix A

ZOSPI

A.1 Visualization of Q-Landscape

Figure A.1 shows the visualization of learned policies (actions given different states) and Q values in TD3 during training in the Pendulum-v0 environment, where the state space is 3-dim and action space is 1-dim. The red lines indicates the selected action by the current policy. The learned Q function are always non-convex, as a consequence, in many states the TD3 is not able to find globally optimal solution and locally gradient information may be misleading in finding actions with high Q values.

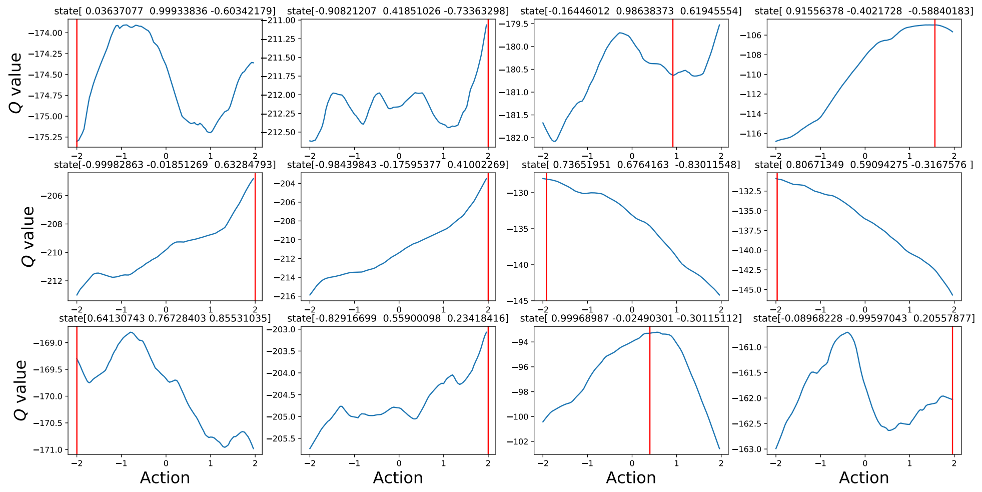


Figure A.1: Landscape of learned value function in the Pendulum-v0 environment

A.2 One-step Zeroth-Order Optimization with Consistent Iteration

Algorithm 5 One-step Zeroth-Order Optimization with Consistent Iteration

```
Require
Objective function $Q$, domain $\mathcal{A}$, current point $a_0$, number of local samples $n_1$, number of global samples $n_2$, local scale $\eta > 0$ and step size $h$, number of steps $m$.
for $t = 1, \dots n_2$ do
    Globally sampling
    Sample a point uniformly in the entire space by
        $$a_{t0} \sim \mathcal{U}_{\mathcal{A}}$$
    where $\mathcal{U}_{\mathcal{A}}$ is the uniform distribution over $\mathcal{A}$.
    for $i = 1, \dots, m$ do
        Locally sampling
        Sample $n_1$ points around $a_{t,i-1}$ by
            $$\tilde{a}_j = a_{t,i-1} + \eta e_j \text{ for } e_j \sim \mathcal{N}(0, I_d), j = 1, \dots n_1,$$
        where $\mathcal{N}(0, I_d)$ is the standard normal distribution centered at 0.
        Update
        Set $a_{t,i} = a_{t,i-1} + h(_{a \in \{\tilde{a}_j\}} Q(a) - a_{t,i-1})$
    end for
end for
return $\max_{a \in \{a_{t,m}\}_{t=1}^{n_2}} Q(a)$.
```

A.3 Proof of Proposition 1

For simplicity, we let $Q(a)$ denote $Q_{w_t}(s_t, a)$. By Taylor expansion, we have $Q(a_i) - Q(a_0) = \nabla Q(a_0)^T a_i'$ for some a_i' between a_i and a_0. Then $_i Q(a_i) =_i \nabla Q(a_0)^T a_i'$.

A.4 Convergence for Zeroth-order Optimization

Proof Sketch. As shown in nesterov2017random, under the same condition in Definition 3.3.4, given $x_0 \dots, x_N \in \mathcal{D}$, we have

$$F(x_N) - F(x^*) \leq \tfrac{\beta}{2}(1 - \tfrac{\alpha}{8(d+4)\beta})^N \|x_0 - x^*\|^2.$$

Thus, as long as there is a global sample lies in $\mathcal{D}$, it requires at most

$$N_\epsilon = \log(\frac{\beta\|x_0 - x^*\|^2}{2\epsilon})\frac{8(d+4)\beta}{\alpha}$$

iterations to find an ϵ-optimal maxima.

The probability of sampling a point in $\mathcal{D}$ globally is at least $\frac{c}{d}$. On expectation, it requires $\frac{d}{c}$ global samples to start from a point in $\mathcal{D}$. Theorem 3.3.4 follows.

A.5 Implementation Details

A.6 Algorithm 6: ZOSPI with Bootstrapped Q networks

Algorithm 6 ZOSPI with UCB Exploration

```
Require
  • The number of epochs $M$, the size of mini-batch $N$, momentum $\tau > 0$ and the number of Bootstrapped Q-networks $K$.
  • Random initialized policy network $\pi_{\theta_1}$, target policy network $\pi_{\theta'_1}$, $\theta'_1 \leftarrow \theta_1$.
  • $K$ random initialized $Q$ networks, and corresponding target networks, parameterized by $w_{k,1}, w'_{k,1}$, $w'_{k,1} \leftarrow w_{k,1}$ for $k = 1, \dots, K$.
for iteration $= 1, 2, ...$ do
  for t $= 1, 2, ..., T$ do
    # Interaction
    Run policy $\pi_{\theta'_t}$, and collect transition tuples $(s_t, a_t, s'_t, r_t, m_t)$.
    for epoch $j = 1, 2, ..., M$ do
      Sample a mini-batch of transition tuples $\mathcal{D}_j = \{(s, a, s', r, m)_i\}_{i=1}^N$.
      # Update $Q$
      for $k = 1, 2, ..., K$ do
        Calculate the $k$-th target $Q$ value $y_{ki} = r_i + \max_l Q_{w'_{k,t}}(s'_i, a'_l)$, where $a'_l \sim \mathcal{U}_{\mathcal{A}}$.
        Update $w_{k,t}$ with loss $\sum_{i=1}^N m_{ik}(y_{ki} - Q_{w'_{k,t}}(s_i, a_i))^2$.
      end for
      # Update $\pi$
      Calculate the predicted action $a_0 = \pi_{\theta'_t}(s_i)$
      Sample actions $a_l \sim \mathcal{U}_{\mathcal{A}}$
      Select $a^+ \in \{a_l\} \cup \{a_0\}$ as the action with maximal $Q^+(s_t, a)$ defined in (3.20).
      Update policy network with Eq.(3.18).
    end for
    $\theta'_{t+1} \leftarrow \tau\theta_t + (1-\tau)\theta'_t$.
    $w'_{k,t+1} \leftarrow \tau w_{k,t} + (1-\tau)w'_{k,t}$.
    $w_{k,t+1} \leftarrow w_{k,t}$; $\theta_{t+1} \leftarrow \theta_t$.
  end for
end for
```

Bibliography

[1] Abbas Abdolmaleki et al. "Relative entropy regularized policy iteration". In: *arXiv preprint arXiv:1812.02256* (2018).

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. "Striving for simplicity in off-policy deep reinforcement learning". In: *arXiv preprint arXiv:1907.04543* (2019).

[3] Larbi Alili, Pierre Patie, and Jesper Lund Pedersen. "Representations of the first hitting time density of an Ornstein-Uhlenbeck process". In: *Stochastic Models* 21.4 (2005), pp. 967–980.

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, et al. "Hindsight Experience Replay". In: *NeurIPS*. 2017, pp. 5048–5058.

[5] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *RAS* (2009).

[6] Himani Arora et al. "Multi-task learning for continuous control". In: *arXiv:1802.01034* (2018).

[7] Christopher G Atkeson and Stefan Schaal. "Robot learning from demonstration". In: *ICML*. Vol. 97. Citeseer. 1997, pp. 12–20.

[8] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. "Minimax regret bounds for reinforcement learning". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 263–272.

[9] Qinbo Bai, Mridul Agarwal, and Vaneet Aggarwal. "Escaping Saddle Points for Zeroth-order Non-convex Optimization using Estimated Gradient Descent". In: *2020 54th Annual Conference on Information Sciences and Systems (CISS)*. IEEE. 2020, pp. 1–6.

[10] Andrew G Barto and Sridhar Mahadevan. "Recent advances in hierarchical reinforcement learning". In: *DEDS* (2003).

[11] Yoshua Bengio et al. "Curriculum learning". In: *ICML*. 2009, pp. 41–48.

[12] Christopher M Bishop. "Mixture density networks". In: (1994).

[13] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[14] Ronen I Brafman and Moshe Tennenholtz. "R-max-a general polynomial time algorithm for near-optimal reinforcement learning". In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231.

[15] Yuri Burda et al. "Exploration by random network distillation". In: *arXiv:1810.12894* (2018).

[16] Yuri Burda et al. "Large-Scale Study of Curiosity-Driven Learning". In: (2018).

[17] Víctor Campos, Xavier Giro-i-Nieto, and Jordi Torres. "Importance Weighted Evolution Strategies". In: *arXiv:1811.04624* (2018).

[18] Kamil Ciosek et al. "Better Exploration with Optimistic Actor Critic". In: *Advances in Neural Information Processing Systems*. 2019, pp. 1785–1796.

[19] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, et al. "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents". In: *NeurIPS*. 2018.

[20] Wojciech Marian Czarnecki, Razvan Pascanu, Simon Osindero, et al. "Distilling policy distillation". In: *arXiv:1902.02186* (2019).

[21] Thomas Degris, Martha White, and Richard S Sutton. "Off-policy actor-critic". In: *arXiv preprint arXiv:1205.4839* (2012).

[22] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[23] Thomas G Dietterich. "Hierarchical reinforcement learning with the MAXQ value function decomposition". In: *JAIR* 13 (2000), pp. 227–303.

[24] Yaakov Engel, Shie Mannor, and Ron Meir. "Reinforcement learning with Gaussian processes". In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 201–208.

[25] Ying Fan, Letian Chen, and Yizhou Wang. "Efficient Model-Free Reinforcement Learning Using Gaussian Process". In: *arXiv preprint arXiv:1812.04359* (2018).

[26] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *arXiv:1802.09477* (2018).

[27] Dibya Ghosh, Abhishek Gupta, Justin Fu, et al. "Learning To Reach Goals Without Reinforcement Learning". In: *arXiv:1912.06088* (2019).

[28] Robert Grande, Thomas Walsh, and Jonathan How. "Sample efficient reinforcement learning with gaussian processes". In: *International Conference on Machine Learning*. 2014, pp. 1332–1340.

[29] Shixiang Gu et al. "Q-prop: Sample-efficient policy gradient with an off-policy critic". In: *arXiv preprint arXiv:1611.02247* (2016).

[30] Tuomas Haarnoja et al. "Reinforcement learning with deep energy-based policies". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1352–1361.

[31] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *arXiv preprint arXiv:1801.01290* (2018).

[32] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[33] Robert Hecht-Nielsen. "Kolmogorov's mapping neural network existence theorem". In: *Proceedings of the IEEE International Conference on Neural Networks III*. IEEE Press. 1987, pp. 11–13.

[34] Todd Hester, Matej Vecerik, Olivier Pietquin, et al. "Deep q-learning from demonstrations". In: *AAAI*. 2018.

[35] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[36] Thomas Jaksch, Ronald Ortner, and Peter Auer. "Near-optimal regret bounds for reinforcement learning". In: *Journal of Machine Learning Research* 11.Apr (2010), pp. 1563–1600.

[37] Chi Jin et al. "Is q-learning provably efficient?" In: *Advances in Neural Information Processing Systems*. 2018, pp. 4863–4873.

[38] Michael I Jordan and David E Rumelhart. "Forward models: Supervised learning with a distal teacher". In: *Cognitive science* 16.3 (1992), pp. 307–354.

[39] Leslie Pack Kaelbling. "Learning to achieve goals". In: *IJCAI*. Citeseer. 1993, pp. 1094–1099.

[40] Aviral Kumar et al. "Stabilizing off-policy q-learning via bootstrapping error reduction". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11761–11771.

[41] Malte Kuss. "Gaussian process models for robust regression, classification, and reinforcement learning". PhD thesis. echnische Universität Darmstadt Darmstadt, Germany, 2006.

[42] Malte Kuss and Carl E Rasmussen. "Gaussian processes in reinforcement learning". In: *Advances in neural information processing systems*. 2004, pp. 751–758.

[43] Vera Kuurkova. "Kolmogorov's theorem and multilayer neural networks". In: *Neural networks* 5.3 (1992), pp. 501–506.

[44] Sergey Levine, Zoran Popovic, and Vladlen Koltun. "Nonlinear inverse reinforcement learning with gaussian processes". In: *Advances in Neural Information Processing Systems*. 2011, pp. 19–27.

[45] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, et al. "Continuous control with deep reinforcement learning". In: *arXiv:1509.02971* (2015).

[46] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple random search provides a competitive approach to reinforcement learning". In: *arXiv:1803.07055* (2018).

[47] Piotr Mirowski et al. "Learning to Navigate in Complex Environments". In: *International Conference on Learning Representations*. 2017.

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[49] Ashvin Nair et al. "Overcoming exploration in reinforcement learning with demonstrations". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6292–6299.

[50] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. 1999.

[51] Ian Osband, John Aslanides, and Albin Cassirer. "Randomized prior functions for deep reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8617–8629.

[52] Ian Osband et al. “Deep exploration via bootstrapped DQN”. In: *Advances in neural information processing systems*. 2016, pp. 4026–4034.

[53] Jakub Pachocki et al. “OpenAI Five, 2018”. In: *URL https://blog. openai. com/openai-five* ().

[54] Deepak Pathak et al. “Curiosity-Driven Exploration by Self-Supervised Prediction”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. July 2017.

[55] Matthias Plappert, Marcin Andrychowicz, Alex Ray, et al. “Multi-goal reinforcement learning: Challenging robotics environments and request for research”. In: *arXiv:1802.09464* (2018).

[56] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, et al. “Parameter space noise for exploration”. In: *arXiv:1706.01905* (2017).

[57] Matthias Plappert et al. “Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research”. In: (2018).

[58] Paulo Rauber et al. “Hindsight policy gradients”. In: *arXiv:1711.06006* (2017).

[59] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, et al. “Policy distillation”. In: *arXiv:1511.06295* (2015).

[60] Tim Salimans, Jonathan Ho, Xi Chen, et al. “Evolution strategies as a scalable alternative to reinforcement learning”. In: (2017).

[61] Stefan Schaal. “Learning from demonstration”. In: *NeurIPS*. 1997, pp. 1040–1046.

[62] Tom Schaul et al. “Universal Value Function Approximators”. In: *ICML*. 2015, pp. 1312–1320.

[63] Juergen Schmidhuber. “Reinforcement Learning Upside Down: Don’t Predict Rewards–Just Map Them to Actions”. In: *arXiv:1912.02875* (2019).

[64] Simon Schmitt, Jonathan J Hudson, Augustin Zidek, et al. “Kickstarting deep reinforcement learning”. In: *arXiv:1803.03835* (2018).

[65] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv:1707.06347* (2017).

[66] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. 2015, pp. 1889–1897.

[67] Evan Shelhamer et al. “Loss is its own reward: Self-supervision for reinforcement learning”. In: *arXiv:1612.07307* (2016).

[68] David Silver et al. “Deterministic policy gradient algorithms”. In: *ICML*. 2014.

[69] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, et al. “Training Agents using Upside-Down Reinforcement Learning”. In: *arXiv:1912.02877* (2019).

[70] Hao Sun et al. “Policy Continuation with Hindsight Inverse Dynamics”. In: *NeurIPS*. 2019.

[71] Richard S Sutton and Andrew G Barto. “Reinforcement Learning: An Introduction”. In: (1998).

[72] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *NeurIPS*. 2000, pp. 1057–1063.

[73] Aviv Tamar et al. “Value iteration networks”. In: *NeurIPS*. 2016, pp. 2154–2162.

[74] Yee Teh, Victor Bapst, Wojciech M Czarnecki, et al. "Distral: Robust multitask reinforcement learning". In: *NeurIPS*. 2017.

[75] Chen Tessler, Guy Tennenholtz, and Shie Mannor. "Distributional Policy Optimization: An Alternative Approach for Continuous Control". In: *arXiv preprint arXiv:1905.09855* (2019).

[76] Nicolas Usunier et al. "Episodic exploration for deep deterministic policies for StarCraft micromanagement". In: (2016).

[77] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[78] Emmanouil-Vasileios Vlatakis-Gkaragkounis, Lampros Flokas, and Georgios Piliouras. "Efficiently avoiding saddle points with zero order methods: No gradients required". In: *Advances in Neural Information Processing Systems*. 2019, pp. 10066–10077.

[79] Qing Wang et al. "Exponentially weighted imitation learning for batched historical data". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6288–6297.

[80] Ziyu Wang et al. "Sample efficient actor-critic with experience replay". In: *arXiv preprint arXiv:1611.01224* (2016).

[81] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[82] Chuheng Zhang, Yuanqi Li, and Jian Li. "Policy Search by Target Distribution Learning for Continuous Control". In: *arXiv:1905.11041* (2019).