Learning to Explore via Meta-Policy Gradient

Tianbing Xu [1] **Qiang Liu** [2] **Liang Zhao** [1] **Jian Peng** [3]

Abstract

The performance of off-policy learning, including deep Q-learning and deep deterministic policy gradient (DDPG), critically depends on the choice of the exploration strategy. Existing exploration methods are mostly based on adding noises to the on-going actor policy and therefore only explore *locally* close to what the actor policy dictates. In this work, we develop a simple meta-policy gradient algorithm that allows us to adaptively learn the exploration policy in DDPG. Our algorithm allows us to train flexible exploration behaviors that are independent of the actor policy, yielding a more *global exploration* that significantly accelerates Q-learning. With an extensive study, we show that our method significantly improves the sample-efficiency of DDPG on a variety of reinforcement learning continuous control tasks.

1. Introduction

Recent advances in deep reinforcement learning (RL) have demonstrated significant applicability and strong performance in games (Mnih et al., 2015; Silver et al., 2017), continuous control (Lillicrap et al., 2016), and robotics (Levine et al., 2016). Among them, deep neural networks, such as convolutional neural networks, are widely used as powerful functional approximators for extracting useful features and enabling complex decision making. For instance, in continuous control tasks, a policy that selects actions under certain state observation can be parameterized with a deep neural network that takes the current state observation as input and gives an action or a distribution of action as output. In order to optimize such policies, various policy gradient methods (Heess et al., 2017; Mnih et al., 2016; Schulman et al., 2015; 2017), including both off-policy and on-policy approaches, have been proposed. In particular, deterministic policy gradient method (DPG), which extends the discrete Q-learning algorithm to the continuous action spaces, exploits previous experience or off-policy data from a replay buffer and often achieves more desirable sample efficiency compared to most existing on-policy policy gradient algorithms. In the recent NIPS 2017 learning to run challenge, the deep deterministic policy gradient algorithm (DDPG) (Lillicrap et al., 2016), a variant of DPG, has been applied by almost all top-ranked teams and achieved a very compelling success in a high-dimensional continuous control problem, while on-policy algorithms, including TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017), performed much worse with the same amount of data collected.

In contrast to deep Q-learning (DQN) (Mnih et al., 2015) which only learns a value function on a set of discrete actions, DDPG also parameterizes a deterministic policy to select a continuous action, thus avoiding the optimization in or the discretization of the continuous action space. As an off-policy actor-critic method, DDPG utilizes the Bellman equation updates for optimizing the value function and the policy gradient method to optimize the actor policy. Unlike DQN which often applies epsilon-greedy exploration on a set of discrete actions, more sophisticated continuous exploration in the high-dimensional continuous action space is required for DDPG. A common practice of exploration in DDPG is to add a uncorrelated Gaussian or a correlated Ornstein-Uhlenbeck (OU) process (Uhlenbeck & Ornstein, 1993) to the action selected by the deterministic policy. The data collected by this exploration method is then added to a replay buffer used for DDPG training. However, in practice, Gaussian noises may be sub-optimal or misspecified, and hyper-parameters in the noise process are hard to tune.

In this work, we introduce a meta-learning algorithm to directly learn an exploration policy to collect better experience data for DDPG training. Instead of using additive noises on actions, we parameterize a stochastic policy to generate data to construct the replay buffer for training the deterministic policy in the DDPG algorithm. This stochastic policy can be seen as an exploration policy or a teacher policy that gathers high-quality trajectories that enable better training of the current deterministic policy and the value function. To learn the exploration policy, we develop an on-policy policy gradient algorithm based on the training improvement of the deterministic policy. First, we obtain a collection of exploration data from the stochastic policy and then apply

[1]Baidu Research, Sunnyvale, CA [2]University of Texas at Austin, TX [3]University of Illinois at Urbana Champaign, IL. Correspondence to: Tianbing Xu <xutianbing@baidu.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

DDPG on this data-set to make updates of the value function and the deterministic policy. We then evaluate the updated deterministic policy and compute the improvement of these updates based on the data just collected by comparing to the previous policy. Therefore, the policy gradient of the stochastic policy can be computed using the deterministic policy improvement as the reward signal. This algorithm adaptively adjusts the exploration policy to generate effective training data for training the deterministic policy. We have performed extensive experiments on several classic control and Mujoco (Todorov et al., 2012) tasks, including Hopper, Reacher, Half-Cheetah, Inverted Pendulum, Inverted Double Pendulum and Pendulum. Compared to the default DDPG in OpenAI's baseline (Plappert et al., 2017), our algorithm demonstrated substantial improvements in terms of sample efficiency. We also compared the default Gaussian exploration and the learned exploration policy and found that the exploration policy tends to visit novel states that are potentially beneficial for training the target deterministic policy.

2. Related Work

The idea of meta learning (Andrychowicz et al., 2016; Bengio et al., 1991; Schmidhuber, 1987) has been widely explored in different areas of machine learning, under different names, such as meta reinforcement learning, life-long learning, learning to learn, and continual learning. Some of the recent work in the setting of reinforcement learning includes (Duan et al., 2016; Finn et al., 2017; Wang et al., 2016), to name a few. Our work is related to the idea of learning to learn but instead of learning to optimize hyperparameters (Maclaurin et al., 2015), neureal network (Chen et al., 2017) or loss functions (Houthooft et al., 2018), we hope to generate high quality data to better train reinforcement agents.

Intrinsic rewards such as prediction gain (Bellemare et al., 2016), learning progress (Oudeyer & Kaplan, 2007), compression progress (Schmidhuber, 2010), variational information maximization (Houthooft et al., 2016; Todd & Peter, 2017), have been employed to augment the environment's reward signal for encouraging to discover novel behavior patterns. One of limitations of these methods is that the intrinsic reward weighting relative to the environment reward must be chosen manually, rather than learned on the fly from interaction with the environment. Another limitation is that the reshaped reward might not guarantee the learned policy to be the same optimal one as that learned from environment rewards only (Ng et al., 1999).

The problem of exploration has been widely used in the literature. Beyond the traditional studies based on epsilon-greedy and Boltzmann exploration (Sutton & Barto, 1998), there are several recent advances in the setting of deep reinforcement learning. For example, (Tang et al., 2017) studied count-based exploration for deep reinforcement learning; (Stadie et al., 2015) proposed a new exploration method based on assigning exploration bonuses from a concurrently learned transition model; (Hester et al., 2013) studied a bandit-based algorithm for learning simple exploration strategies in model-based settings; (Osband et al., 2016a) used a bootstrapped approach for exploration in DQN, a simple algorithm in a computationally and statistically efficient manner through the use of randomized value functions (Osband et al., 2016b).

3. Reinforcement learning

In this section, we introduce the background of reinforcement learning. We start with introducing Q-learning in Section 3.1, and then deep deterministic policy gradient (DDPG) which works for continuous action spaces in Section 3.2.

3.1. Q-learning

Considering the standard reinforcement learning setting, an agent takes a sequence of actions in an environment in discrete time and collects a scalar reward per timestep. The objective of reinforcement learning is to learn a policy of the agent to optimize the cumulative reward over future time. More precisely, we consider an agent act over time $t \in \{1, \dots, T\}$. At time t, the agent observes an environment state s_t and selects an action $a_t \in A$ to take according to a policy. The policy can be either a deterministic function $a = \mu(s)$, or more generally a conditional probability $\pi(a|s)$. The agent will then observe a new state s_{t+1} and receive a scalar reward value $r_t \in R$. The set A of possible actions can be discrete, continuous or mixed in different tasks. Given a trajectory $\{s_t, a_t, r_t\}_{t=1}^T$, the overall reward is defined as a discounted sum of incremental rewards, $R = \sum_{t=1}^T \gamma^t r_t$, where $\gamma \in [0, 1)$ is a discount factor. The goal of RL is to find the optimal policy to maximize the expected reward.

Q-learning (Watkins, 1989; Watkins & Dayan, 1992) is a well-established method that has been widely used. Generally, Q-learning algorithms compute an action-value function, often also referred to as Q-function, $Q^*(s, a)$, which is the expected reward of taking a given action a in a given state s, and following an optimal policy thereafter. The estimated future reward is computed based on the current state s or a series of past states if available.

The core idea of Q-learning is the use of the Bellman equation as a characterization of the optimal future reward function Q^* via a state-action-value function

$$Q^*(s_t, a) = \mathbb{E}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')], \qquad (1)$$

where the expectation is taken w.r.t the distribution of state s_{t+1} and reward r_t obtained after taking action a. Given the optimal Q-function, the optimal policy greedily selects the actions with the best Q-function values. Deep Q-learning (DQN), a recent variant of Q-learning, uses deep neural networks as Q-function to automatically extract intermediate features from the state observations and shows good performance on various complex high-dimensional tasks.

Since Q-learning is off-policy, a particular technique called "experience replay" (Lin, 1992; Wawrzyski, 2009) that stores past observations from previous trajectories for training has become a standard step in deep Q-learning. Experience replays are stored as a dataset, also known as replay buffer, $B = \{(s_j, a_j, r_j, s_{j+1})\}$ which contains a set of previously observed state-action-reward-future state-tuples (s_j, a_j, r_j, s_{j+1}). Such experience replays are often constructed by pooling such tuples generated by recent policies.

With the replay buffer B, Deep Q learning follows the following iterative procedure (Mnih et al., 2013; 2015): start an episode in the initial state s_0; sample a mini-batch of tuples $M = \{(s_j, a_j, r_j, s_{j+1})\} \subseteq B$; compute and fix the targets $y_j = r_j + \gamma \max_a Q_{\theta^-}(s_{j+1}, a)$ for each tuple using a recent estimate Q_{θ^-} (the maximization is only considered if s_j is not a terminal state); update the Q-function by optimizing the following program the parameters θ typically via stochastic gradient descent:

$$\min_{\theta} \sum_{(s_j, a_j, r_j, s_{j+1}) \in M} \left(Q_\theta(s_j, a_j) - y_j\right)^2. \qquad (2)$$

Besides updating the parameters of the Q-function, each step of Q-learning needs to gather additional data to augment the replay buffer. This is done by performing an action simulation either by choosing an action at random with a small probability ϵ or by following the strategy $\arg\max_a Q_\theta(s_t, a)$ which is currently estimated. This strategy is also called the ϵ-greedy policy which is applied to encourage visiting unseen states for better exploration and avoid the training stuck at some local minima. We subsequently obtain the reward r_t. Subsequently we augment the replay buffer B with the new tuple (s_t, a_t, r_t, s_{t+1}) and continue until this episode terminates or reaches an upper bound of timesteps, and then we restart a new episode. When optimizing the parameter θ, a recent Q-network is used to compute the target $y_j = r_j + \gamma \max_a Q_{\theta^-}(s_{j+1}, a)$.

3.2. Deep Deterministic Policy Gradient

For continuous action spaces, it is practically impossible to directly apply Q-learning, because the max operator in the Bellman equation, which find the optimal a, is usually infeasible, unless discretization is used or some special forms of Q-function are used. Deep deterministic policy gradient (DDPG) (Lillicrap et al. (2016)) addresses this issue by training a parametric policy network together with the Q-function using policy gradient descent.

Specifically, DDPG maintains a deterministic actor policy $\pi = \delta(a - \mu(s, \theta^\pi))$ where $\mu(s, \theta^\pi)$ is a parametric function, such as a neural network, that maps the state to actor. We want to iteratively update θ^π, such that $a = \mu(s, \theta^\pi)$ gives the optimal action that maximizes the Q-function $Q(s, a)$. so that $a = \mu(s, \theta^\pi)$ can be viewed as an approximate action-argmax operator of the Q-function, and we do not have to perform the action maximization in the high-dimensional continuous space. In training, the critic $Q_\theta(s, a)$ is updated using the Bellman equation as in Q-learning that we introduced above, and the actor is updated to maximize the expected reward w.r.t. $Q_\theta(s, a)$,

$$\max_{\theta^\pi} \left\{ J(\theta^\pi) := \mathbb{E}_{s \sim B}[Q_\theta(s, \mu(s, \theta^\pi))] \right\},$$

where $s \sim B$ denotes sampling s from the replay buffer B. This is achieved in DDPG using the gradient update:

$$\theta^\pi \leftarrow \theta^\pi + \nabla_{\theta^\pi} J(\theta^\pi),$$

where,

$$\nabla_{\theta^\pi} J(\theta^\pi) = \mathbb{E}_{s \sim B}[\nabla_a Q_\theta(s, \mu(s, \theta^\pi)) \nabla_{\theta^\pi} \mu(s)]$$

In DDPG, the actor $\mu(s, \theta^\pi)$ and the critic $Q_\theta(s, a)$ are updated alternatively until convergence.

As in Q-learning, the performance of DDPG critically depends on a proper choice of exploration policy π_e, which controls what data to add at each iteration. However, in high-dimensional continuous action space, exploration is highly nontrivial. In the current practice of DDPG, the exploration policy π_e is often constructed heuristically by adding certain type of noise to the actor policy to encourage stochastic exploration. A common practice is to add an uncorrelated Gaussian or a correlated Ornstein-Uhlenbeck (OU) process (Uhlenbeck & Ornstein, 1993) to the action selected by the deterministic actor policy. Since DDPG is off-policy, the exploration can be independently addressed from the learning. It is still unclear whether these exploration strategies can always lead to desirable learning of the deterministic actor policy.

4. Learning to Explore

We expect to construct better exploration strategies that are potentially better than the default Gaussian or OU exploration. In practice, e.g., in the Mujuco control tasks, the action spaces are bounded by a high-dimensional continuous cube $[-1, 1]^d$. Therefore, it is very possible that the Gaussian assumption of the exploration noises is not suitable when the action selected by the actor policy is close to the corner or boundaries of this cube. Furthermore it is

Algorithm 1 Teacher: Learn to Explore

1: Initialize π_e and π.
2: Draw D_1 from π to estimate the reward $\hat{R}_\pi$ of π.
3: Initialize the Replay Buffer $B = D_1$.
4: **for** iteration t **do**
5: Generate D_0 by executing teacher's policy π_e.
6: Update actor policy π to π' using DDPG based on D_0: $\pi' \leftarrow \mathrm{DDPG}(\pi, D_0)$.
7: Generate D_1 from π' and estimate the reward of π'. Calculate the meta reward: $\hat{\mathcal{R}}(\pi, D_0) = \hat{R}_{\pi'} - \hat{R}_\pi$.
8: Update Teacher's Policy π_e with meta policy gradient

$$\theta^{\pi_e} \leftarrow \theta^{\pi_e} + \eta \nabla_{\theta^{\pi_e}} \log \mathcal{P}(D_0|\pi_e) \hat{\mathcal{R}}(\pi, D_0)$$

9: Add both D_0 and D_1 into the Replay Buffer $B \leftarrow B \bigcup D_0 \bigcup D_1$.
10: Update π using DDPG based on Replay Buffer, that is, $\pi \leftarrow \mathrm{DDPG}(\pi,\ B)$. Compute the new $\hat{R}_\pi$.
11: **end for**

also possible that the actor policy gets stuck in a local basin in the state space and thus cannot escape even with random Gaussian noises added.

All existing exploration strategies seem to be based on the implicit assumption that the exploration policy π_e should stay close to the actor policy π, but with some more stochastic noise. However, this assumption may not be true. Instead, it may be beneficial to make π_e significantly different from the actor π in order to explore the space that has not been explored previously. Even in the case of using Gaussian noise for exploration, the magnitude of the Gaussian noise is also a critical parameter that may influence the performance significantly. Therefore, it is of great importance to develop a systematic approach to adaptively learn the exploration strategy, instead of using simple heuristics.

Since DDPG is an off-policy learning algorithm and the exploration is independent from the learning, we can decouple the exploration policy with the actor policy. We hope to construct an exploration policy which generates novel experience replays that are more beneficial for training the actor policy. To do so, we introduce a meta-reinforcement learning approach to learn an exploration policy so that it most efficiently improves the training of the actor policy.

4.1. Learning Exploration Policy with Policy Gradient

Our framework can be best viewed as a teacher-student learning framework, where the *teacher*'s exploration policy π_e, generates a set of data D_0 at each iteration, and feeds it into a *student* agent with an actor (or exploitation) policy π, who learns from the data and improves itself. Our goal is to adaptively improve the teacher π_e so that it generates the most informative data to make the DDPG learner improve as fast as possible.

In this meta framework, the generation of data D_0 can be viewed as the "action" taken by the teacher (with policy π_e), and its related reward should be defined as the improvement of the student with DDPG learner using data D_0,

$$\begin{aligned} \mathcal{J}(\pi_e) &= \mathbb{E}_{D_0 \sim \pi_e}[\mathcal{R}(\pi, D_0)] \\ &= \mathbb{E}_{D_0 \sim \pi_e}[R_{\pi'} - R_\pi], \end{aligned} \tag{3}$$

where $\pi' = \mathrm{DDPG}(\pi, D_0)$ denotes a new policy obtained from one or a few steps of DDPG updates from π based on data D_0; $R_{\pi'}$ and R_π are the actual cumulative reward of rollouts generated by policies π' and π, respectively, in the original RL problem. Here the **meta-reward** $\mathcal{R}(\pi, D_0)$ denotes how much the teacher helps the progress of student's learning.

Similar to the actor policy, we can parameterize this exploration policy π_e by θ^{π_e}. Using the REINFORCE trick, we can calculate the gradient of $\mathcal{J}(\pi_e)$ w.r.t. θ^{π_e}:

$$\nabla_{\theta^{\pi_e}} \mathcal{J} = \mathbb{E}_{D_0 \sim \pi_e} \left[\mathcal{R}(\pi, D_0) \nabla_{\theta^{\pi_e}} \log \mathcal{P}(D_0|\pi_e) \right], \tag{4}$$

where $\mathcal{P}(D_0|\pi_e)$ is the probability of generating transition tuples $D_0 := \{s_t, a_t, r_t\}_{t=1}^T$ given π_e. This distribution can be factorized as

$$\mathcal{P}(D_0|\pi_e) = p(s_1) \prod_{t=1}^{T} \pi_e(a_t|s_t) p(s_{t+1}|s_t, a_t),$$

where $p(s_{t+1}|s_t, a_t)$ is the transition probability and $p(s_1)$ the initial distribution. The dependency of the reward is omitted here. Because $p(s_{t+1}|s_t, a_t)$ is not involved with the exploration parameter θ^{π_e}, by taking the gradient w.r.t. θ^{π_e}, we have

$$\nabla_{\theta^{\pi_e}} \log \mathcal{P}(D_0|\pi_e) = \sum_{t=1}^{T} \nabla_{\theta^{\pi_e}} \log \pi_e(a_t|s_t).$$

This can be estimated easily on the rollout data. We can also approximate this gradient with sub-sampling for the efficiency purpose.

To estimate the meta-reward $\mathcal{R}(\pi, D_0)$, we perform an "exercise move" by running DDPG ahead for one or a small number of steps: we first calculate a new actor policy $\pi' = \mathrm{DDPG}(\pi, D_0)$ by running DDPG based on data D_0; we then simulate from the new policy π' to get data D_1, and use D_1 to get an estimation $\hat{R}_{\pi'}$ of the reward of π'. This allows us to estimate the meta reward by

$$\hat{\mathcal{R}}(\pi, D_0) = \hat{R}_{\pi'} - \hat{R}_\pi,$$

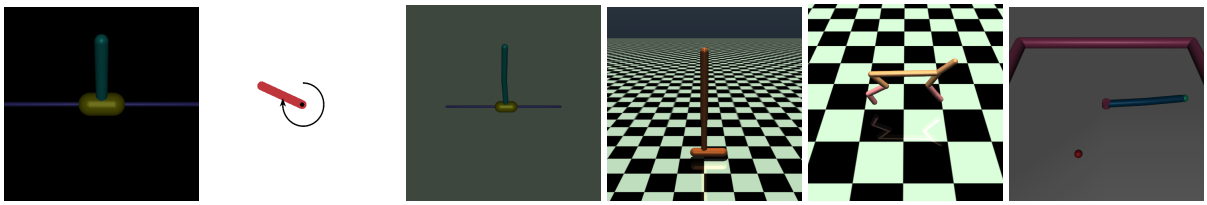


Figure 1. Illustrative screenshots of environments we experiment with Meta and DDPG

where $\hat{R}_\pi$ is the estimated reward of π, which we should have obtained from the previous iteration.

Once we estimate the meta-reward $\mathcal{R}(\pi, D_0)$, we can update the exploration policy π_e by following the meta policy gradient in (4). This yields the following update rule:

$$\theta^{\pi_e} \leftarrow \theta^{\pi_e} + \eta \hat{\mathcal{R}}(\pi, D_0) \sum_{t=1}^{T} \nabla_{\theta^{\pi_e}} \log \pi_e(a_t|s_t). \quad (5)$$

After updating the exploration policy, we add both D_0 and D_1 into a replay buffer B that we maintain across the whole process, that is, $B \leftarrow B \cup D_0 \cup D_1$; we then update the actor policy π based on B, that is, $\pi \leftarrow \text{DDPG}(\pi,\ B)$. Our main algorithm is summarized in Algorithm 1.

It may appear that our meta update adds significant computation demand, especially in requiring to generate D_1 for the purpose of evaluation. However, D_1 is used highly efficiently since it is also added into the replay buffer and is subsequently used for updating π. This design helps improve, instead of decrease, the sample efficiency.

Our framework allows us to explore different exploration policy π_e in term of parametric forms, architectures and features. We tested three design choices:

i) `Meta(variance)`: Similar to and motivated by the traditional exploration strategy, we can set π_e to equal the actor policy adding a zero-mean Gaussian noise whose variance is trained adaptively, that is, $\pi_e = \mathcal{N}(\mu(s, \theta^\pi),\ \sigma^2 I)$, where σ is viewed as the parameter of π_e and is trained with meta policy gradient (5).

ii) `Meta`: We can also take π_e to be another Gaussian policy that is completely independent of π, that is, $\pi_e = \mathcal{N}(f(s, \theta^f), \sigma^2 I)$, where f is a neural network with parameter θ^f, and $\theta^{\pi_e} := [\theta^f, \sigma]$ is updated by the meta policy gradient (5).

iii) `Meta(state)` : In addition to the basic MDP states in `Meta`, we argument with "algorithmic states" such as the normalized Q function and the Bellman residual error of the Q function between the current and last policies.

5. Experiments

In this section, we conduct comprehensive experiments to understand our proposed meta-exploration-policy learning algorithm and to demonstrate its performance in various continuous control tasks. Videos[1] are included to illustrate the running results of Pendulum and Inverted Double Pendulum.

5.1. Experimental Setting

Our implementation is based on the OpenAI's DDPG baseline (Plappert et al., 2017) GitHub[2]. Our experiments were performed on a server with 8 Tesla-M40-24GB GPU and 40 Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz processors. The deterministic actor (or student) policy network and Q-networks have the same architectures as implemented in the default DDPG baseline, which are multi-layer perceptrons with two hidden layers (64-64). For the meta-exploration policy (`Meta` π_e), we implemented a stochastic Gaussian policy with a mean network represented with a MLP with two hidden layers (64-64), and a log-standard-deviation variance variable.

In order to make a fair comparison with baseline, we try to set the similar hyper-parameters as DDPG in most of the common parameters. The parameter settings are: exploration rollout steps (typically 100) for generating exploration trajectories D_0, number of evaluation steps (typically 200, same as DDPG's rollout steps) for generating exploitation trajectories D_1 used to evaluate student's performance, number of training steps (typically 50, aligning with DDPG's training steps) to update student policy π, and number of exploration training steps (typically 1) to update the Meta policy π_e. In most experiments, we set the number of cycles to be 20 in an epoch to align with DDPG's corresponding setting. Tasks such as Half-Cheetah, Inverted Pendulum, need more explore rollout steps (1000) to finish the task, and ended up with 2000 evaluation steps, 500 number of training steps to update students and 100 exploration training steps to update teacher. In OpenAI's DDPG baseline (Plappert et al., 2017), the total number of steps of interactions is 1 million. Here, the number of steps to achieve convergence is 1.5 million for Half-Cheetah, Inverted Pendulum and Inverted Double Pendulum, 1 million for Hopper, 0.7 million for Reacher, and 0.9 million for Pendulum. Similar to DDPG, the optimizer we use to update the network parameter is

[1] https://bit.ly/2ICsuyU

[2] https://github.com/openai/baselines/tree/master/baselines/ddpg

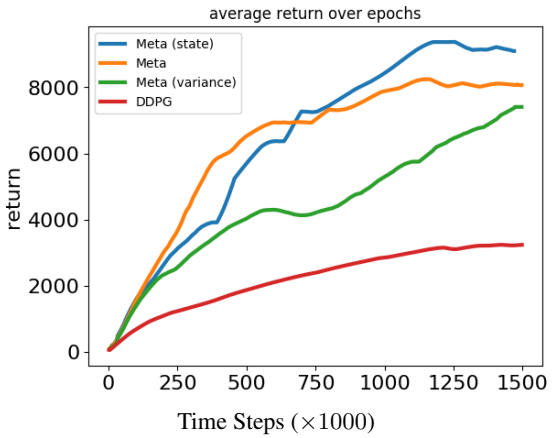


Figure 2. Comparison between meta exploration policies and DDPG

Adam (Kingma & Ba, 2015) with the same actor learning rate 0.0001, critic learning rate 0.001, and additional learning rate 0.0001 for our meta policy. Similar to DDPG, we adopt Layer-Normalization (Ba et al., 2016) for our two policy networks and one Q-network.

5.2. Meta Exploration Policy Explores Efficiently

To investigate and evaluate different teacher's behaviors, we tested in Inverted Double Pendulum the three possible choices of policy designs of π_e listed in Section 4.

In Figure 2, `Meta` denotes that we learn an exploration policy that is a Gaussian MLP policy with a network architecture independent of student's policy. Meta runs consistently better than DDPG baseline with relatively higher return and sample-efficiency. Usually, Meta policy learns in the same pace as student policy, it updates every time both from student's success (performance improvement) and failure (negative performance). For a further more robust policy updates, we may need to take consideration of the trade-off between sample efficiency and sample quality.

A second exploration policy denoted as `Meta (variance)` in Figure 2 is by taking advantage of student's learning, combined with a variance as $\pi_e = \pi + N(0, \sigma^2 I)$. Essentially, we are learning adaptive variance for exploration. Based on the student's performance, teacher is able to learn to provide training transitions with appropriate noise. This teacher's demonstrations help student to explore different regions of state space in an adaptive way.

Table 1. Performance results on six continuous control tasks

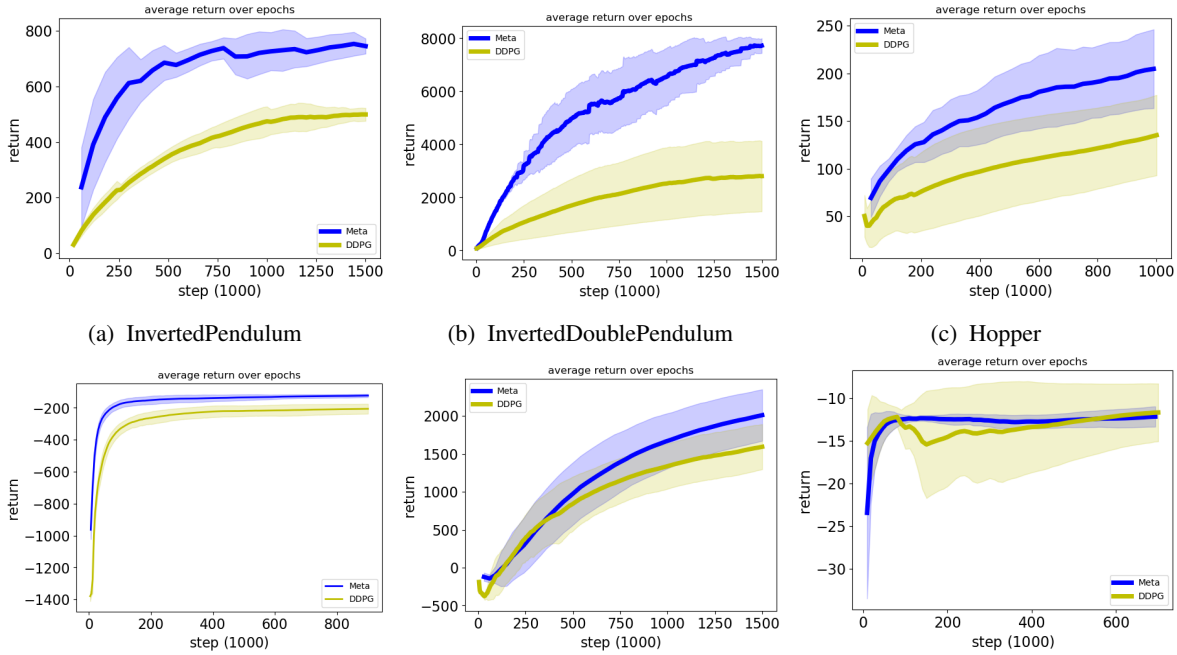
Task	Ours	DDPG
InvertedDoublePendulum	**7718** $\pm$ **277**	2795 $\pm$ 1325
InvertedPendulum	**745** $\pm$ **27**	499 $\pm$ 23
Hopper	**205** $\pm$ **41**	135 $\pm$ 42
Pendulum	**-123** $\pm$ **10**	-206 $\pm$ 31
HalfCheetah	**2011** $\pm$ **339**	1594 $\pm$ 298
Reacher	-12.16 $\pm$ 1.19	**-11.67** $\pm$ **3.39**

For Figure 2 (`Meta`), we can see that the fully independent exploration policy performs better than the more restrictive policy that only adds noise to the action policy. As we show in Figure 4, the independent exploration policy tends to explore regions that are not covered by the actor policy, suggesting that it is beneficial to perform *non-local exploration*.

Furthermore, in Figure 2 (`Meta(state)`), we find that explicitly adding the "algorithmic state" features can further improve the results, though not very significant in this case. This may be because the algorithm itself already has the ability of adjusting the exploration policy adaptively according to the "algorithmic states" via meta-reward (depending on exploitation policy π and 'action' D_0).

5.3. Sample Efficiency in Continuous Control Tasks

We show the learning curves in Figure 3 for six continuous control tasks, each running three times with different random seeds to produce reliable comparison. The x-axis is the number of thousand steps of interactions, the y-axis is the average return. In the most experiments, we maintain a ratio (1:2) of exploration and evaluation trajectories, which ends up with 1.5 times of interactions per epoch compared to that of DDPG baseline. However, due to Meta's effective learning with adaptive and informative exploration trajectories, we are able to achieve better returns with much less number of epochs. Overall, our meta-learning algorithm is able to achieve sample-efficiency with better returns in most of the following continuous control tasks. Significantly, in Inverted Pendulum and Inverted Double Pendulum, on average, in about 250 thousands out of 1500 thousands steps, we are able to achieve the similar return as the best of DDPG. That is about $1/6$ number of baseline's samples. Finally, our average return of Inverted Double Pendulum is about 7718 compared to DDPG's 2795 (see Table 1). In Pendulum, we performed clearly better with higher average return, and converge faster than DDPG in less than 200 thousand steps. In Half-Cheetah and Hopper, on average, our meta-learning algorithm is pretty robust with higher returns and better sample-efficiency. In Reacher, we have very similar return as DDPG baseline with lower variance. The possible intuition that we are able to improve the sample-efficiency and returns in most of tasks is that teacher is able to learn to help student to improve their performance, which is the student's ultimate goal.



(a) InvertedPendulum (b) InvertedDoublePendulum (c) Hopper

(d) Pendulum (e) HalfCheetah (f) Reacher

Figure 3. Performance Comparison of Meta and DDPG for Six Continuous Control Tasks. The x-axis is the number of thousand steps of interactions, the y-axis is the average return.

Table 2. Comparing with non-adaptive exploration policies. "Adaptive" indicates our adaptive meta policy gradient method. "Final" means the trained final exploration policy. "Mixture" means a mixture of exploration polices trained in the last 100 iterations.

Task	Setting	Return
InvertedDoublePendulum	Adaptive	**7718 ± 277**
InvertedDoublePendulum	Final	182 ± 28
InvertedDoublePendulum	Mixture	124 ± 16
Pendulum	Adaptive	**-123 ± 10**
Pendulum	Final	-184 ± 22
Pendulum	Mixture	-202 ± 1.2

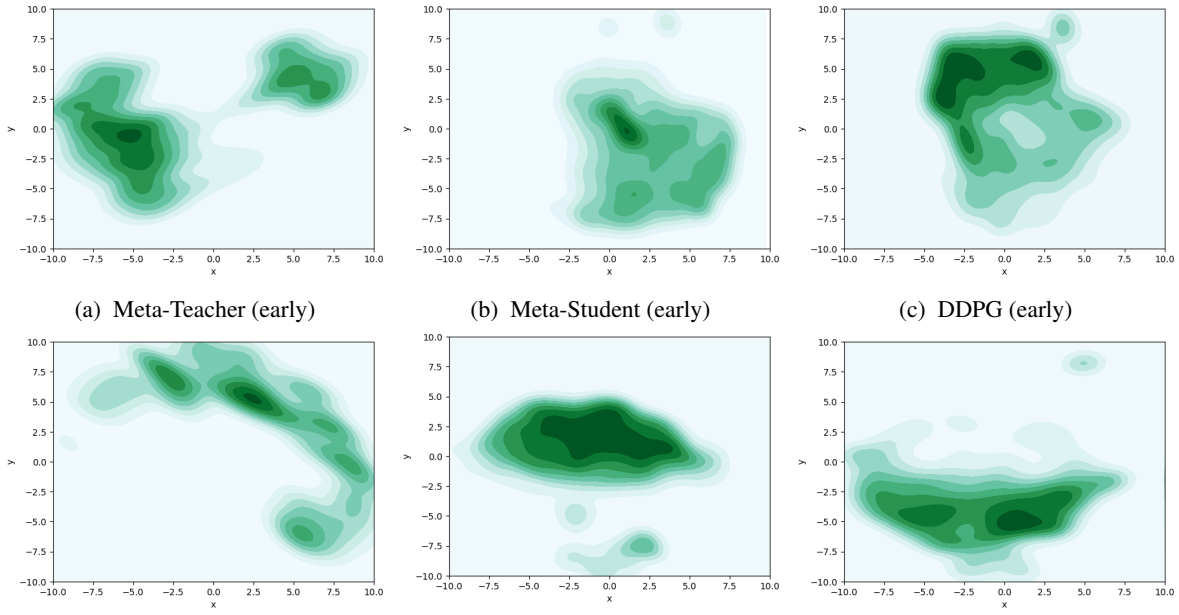
5.4. Guided Exploration with Diverse and Adaptive Meta Policies

To study the importance of adaptiveness of our meta-policy, in comparison, we run DDPG with two fixed and non-adaptive exploration policies: (a) the fixed final exploration policy trained by Meta policy gradient, and (b) a mixture of exploration polices in the last 100 iterations of Meta policy gradient in two environments. We find that both of these cases fail to learn well (see Table 2). This suggests that our meta-exploration does more than finding a best fixed policy as it is able to adaptively adjust the exploration policy more efficiently.

To further understand the behaviors of teacher and student policies and how teacher interacts with student during the learning process, we plot the density contours of state visitation probabilities in Figure 4. The probabilities are learned with Kernel Density Estimation based on the samples in 2D embedding space. In Inverted Double Pendulum task, we collect about 500 thousands observation states for teacher policy and 1 million states for student policy. As comparison, we get 1 million states from DDPG policy. Then we project these data-sets jointly into 2D embedding space by t-SNE (Maaten & Hinton, 2008). We may be able to find interesting insights, although it is possible that the t-SNE projection might introduce artifacts in the visualization.

As shown in Figure 4, we have two groups of comparison studies for the evolution of teacher and student learning processes in different stages. In each row, the first column is Meta-Teacher, the second one is Meta-Student policy and the third one is the DDPG baseline. The first row (Figure 4(a, b, c)) visualizes state distributions from the first 50 roll-outs by executing the random teacher and student policies where the policies are far from becoming stationary. The bottom row (Figure 4(d,e,f)) demonstrates that the state distribution landscape visited by teacher, student and DDPG, respectively, from the last 50 roll-outs to the end of learning.

The teacher is exploring the state space in a *global* way, which visits different regions of state space compared to



(a) Meta-Teacher (early) (b) Meta-Student (early) (c) DDPG (early)

(d) Meta-Teacher (late) (e) Meta-Student (late) (f) DDPG (late)

Figure 4. State Visitation Density Contours of Meta and DDPG in Early and Late Training Stages. In each row, the first column is Meta-Teacher, the second one is Meta-Student policy and the third one is the DDPG baseline.

that of student's state space. In the two learning stages, the Meta-Teacher (Figure 4(a, d)) has diversified state visitation distributions ranging from different modes in separate regions. We can see that Meta-Teacher policy has high entropy, which implies that Meta-Teacher provides more diverse samples for student. Guided by teacher's wide exploration, student policy is able to learn from a large range of state distribution regions.

Interestingly, compared to teacher's behavior, the student visits almost complementary different states in distribution space consistently in both the early (Figure 4(a,b)), and late (Figure 4(d,e)) stages. We can see that the teacher interacts with the student and is able to learn to explore different regions based on student's performance. Meanwhile, the student is learning from teacher's provided demonstrations and is focusing on different regions systematically. This allows the student to improve its performance consistently and continuously. It indicates that our *global exploration* strategy is quite different from noise-based random walk *local exploration* in principle.

From the early (Figure 4(b)) to the late stage (Figure 4(e)), we find that the student is growing to be able to learn stationary and robust policies, guided by teacher's interactive exploration. Finally, compared to DDPG (Figure 4(f)), we achieve better best return (8530 vs 2830) for this comparison, which indicates that our Meta policy is able to provide a better exploration strategy to help improve the baseline.

6. Conclusion and Future Work

We introduce a meta-learning algorithm to adaptively learn exploration polices to collect better experience data for DDPG training. Using a simple meta policy gradient, we are able to efficiently improve the exploration policy and achieve significantly higher sample efficiency than the traditional DDPG training. Our empirical study demonstrates the significant practical advantages of our approach.

Although most traditional exploration techniques are based on *local exploration* around the actor policy, we show that it is possible and more efficient to perform *global exploration*, by training an independent exploration policy that allows us to explore spaces that are far away from the current state distribution. This finding has a substantial implication to our understanding on exploration strategies, showing that more adaptive, non-local methods should be used in order to learn more efficiently. Finally, this meta-policy algorithm is general and could be applied to other off-policy reinforcement learning problems.

Our work has limitations and future research is needed. Our meta-exploration is learned with policy gradient and works well on tasks in continuous control benchmark. However, it is still *heuristic* in the sense of lacking a systematic theoretic analysis. Moreover, it is important to investigate the exploration efficiency of our meta-exploration method on the challenging reinforcement learning tasks in more complex environments with longer time dependencies.

Acknowledgement

We sincerely appreciate the constructive comments from our three anonymous reviewers, which improve our paper significantly. Great thanks to Kliegl Markus for his insightful discussions and comments.

References

Andrychowicz, M., Denil, M., Gmez, S., and et al. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Ba, J. L., Kiros, R., and Hinton, G. E. Layer normalization. *arXiv*, 2016. URL https://arxiv.org/abs/1607.06450.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., and et al. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1471–1479, 2016.

Bengio, Y., Bengio, S., and Cloutier, J. Learning a synaptic learning rule. *Neural Networks, IJCNN-91-Seattle International Joint Conference on*, 1991.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., and et al. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv*, 2016. URL https://arxiv.org/abs/1611.02779.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Heess, N., TB, D., Sriram, S., and et al. Emergence of locomotion behaviours in rich environments. *arXiv*, 2017. URL https://arxiv.org/pdf/1707.02286.pdf.

Hester, T., Lopes, M., and Stone, P. Learning exploration strategies in model-based reinforcement learning. In *Proceedings of the international conference on Autonomous agents and multi-agent systems*, pp. 1069–1076, 2013.

Houthooft, R., Chen, X., Duan, Y., and et al. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1109–1117, 2016.

Houthooft, R., Chen, R. Y., Isola, P., and et al. Evolved policy gradients. *arXiv*, 2018. URL https://arxiv.org/abs/1802.04821.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *Proceedings of the Conference on Learning Representations (ICLR)*, 2015.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, pp. 1334–1373, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., and et al. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, pp. 293321, 1992.

Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Maclaurin, D., Duvenaud, D., and Adams, R. P. Gradient-based hyperparameter optimization through reversible learning. *arXiv*, 2015. URL https://arxiv.org/abs/1502.03492.

Mnih, V., Kavukcuoglu, K., Silver, D., and et al. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., and et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., and et al. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2016.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, 1999.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pp. 2377–2386, 2016a.

Osband, I., Van Roy, B., and Zheng, W. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, pp. 4026–4034, 2016b.

Oudeyer, P.-Y. and Kaplan, F. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1, 2007.

Plappert, M., Houthooft, R., Dhariwal, P., and et al. Parameter space noise for exploration. *arXiv*, 2017. URL https://arxiv.org/abs/1706.01905.

Schmidhuber, J. Evolutionary principles in self-referential learning. *Diploma Thesis, Tech. Univ. Munich*, 1987.

Schmidhuber, J. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv*, 2017. URL https://arxiv.org/abs/1707.06347.

Silver, D., Schrittwieser, J., Simonyan, K., and et al. Mastering the game of go without human knowledge. *Nature*, 550:354359, 2017.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv*, 2015. URL https://arxiv.org/abs/1507.00814.

Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

Tang, H., Houthooft, R., Foote, D., and et al. # Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2750–2759, 2017.

Todd, H. and Peter, S. Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence*, 247: 170–186, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Uhlenbeck, G. E. and Ornstein, L. S. On the theory of the brownian motion. *Physical Review*, 36:823, 1993.

Wang, J. X., Kurth-Nelson, Z., and et al. Learning to reinforcement learn. *arXiv*, 2016. URL https://arxiv.org/abs/1611.05763.

Watkins, C. Learning from delayed rewards. *Ph.D. Thesis, Cambridge*, 1989.

Watkins, C. and Dayan, P. Q learning: Technical note. *Machine Learning*, pp. 279–292, 1992.

Wawrzyski, P. Real-time reinforcement learning by sequential actor-critics and experience replay. *Neural Networks*, pp. 1484–1497, 2009.