Deep Exploration via Stochastic Gradient Langevin Dynamic in Deep Reinforcement Learning

Anonymous Authors[1]

Abstract

Thompson Sampling method can achieve effective exploration in the reinforcement learning, but many existing works fail in sparse/deceptive tasks. The factors of "zero reward" and "action penalty" prevent agents from exploring more states in sparse/deceptive environment, which makes the environment difficult to explore. In this paper, we propose an Thompson sampling exploration strategy based on Stochastic Gradient Langevin Dynamics for Actor-critic algorithm, which can archive exploration in a sparse/deceptive environment and overcomes the shortcomings of existing exploration methods. We measured the performance of our method in the three benchmarks: CountinousMountainCar, SparseHalfCheetah, and HalfCheetah, which represent different attributes. The results show that our approach can solve more tasks, have higher sample efficiency, and lower computational cost. Our code is open source on github.com/anonymousforcodesubmit/SGLD.

1. Introduction

The application of deep learning in reinforcement learning has brought a lot of research progress (Mnih et al., 2015; Silver et al., 2016; OpenAI, 2018). Neural networks can fit arbitrary functions and have many extended forms on different tasks, which enabling reinforcement learning to solve various tasks, such as CNN for pixel input and RNN for non-Markov tasks(Mnih et al., 2015; Bakker, 2002). The latest generation of deep learning frameworks has convenient automatic derivation, integrate various optimizers (Paszke et al., 2017; Chen et al., 2015; Abadi et al., 2016). As in supervised learning, the "loss function + SGD-base optimizer" approach becomes the general way of using neural networks in RL, both for Q-learning or policy gradient (Lillicrap et al., 2015; Mnih et al., 2015; Schulman et al., 2017).

However, in more difficult tasks, especially those with continuous state-action space, or sparse/deceptive rewards, more effective exploration strategies become the key to solving the problem (Plappert et al., 2017; Colas et al., 2018). In the sparse reward task, except for a few specific events, the agent can only get 0 rewards, which will require the agent to effectively traverse the policy space without any incentives (Houthooft et al., 2016). In the deceptive reward task, due to the action penalty in the reward, the agent will receive negative feedback when exploring, which leads the agent to abandon the exploration and maintain the "zero action" (Lehman & Stanley, 2011; Conti et al., 2018b). This situation requires the agent to continue to explore the unknown part of the state action space. In the opposite case, if the task has a well-defined reward, which can guide the agent to learn the optimal strategy, too much exploration will hurt the sample efficiency (Riquelme et al., 2018). The trade-off between exploration and exploitation is an important research issue for reinforcement learning.

There has been a lot of research work on the agent's exploration strategy, including some naive random search methods (action space or parameter space) (Plappert et al., 2017; Lillicrap et al., 2015), count-based methods (Tang et al., 2017; Bellemare et al., 2016), evolutionary strategy (Khadka & Tumer, 2018; Conti et al., 2018a) and so on. There is a class of methods based on Thomson's sampling (Thompson, 1933), the core idea of which is to make decisions using samples generated from the posterior distribution of the value model on the current data. Thompson sampling has an "Instance-Independent Regret bounds" (Russo et al., 2018) and has proven to be effective in a variety of deep learning-based reinforcement learning scenarios (Osband et al., 2016; Houthooft et al., 2016; Henderson et al., 2017; Azizzadenesheli et al., 2018).

However, for an algorithm that uses a neural network as a function of value, it is difficult to directly calculate the posterior distribution. In response to this, there are a series of works to achieve approximate a posteriori sampling using various approximation methods, including dropout (Henderson et al., 2017), variational inference (Houthooft et al.,

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2016), Bootstrap (Osband et al., 2016), and so on. However, the posterior distributions followed by these methods are not accurate, and there may be problems such as lack of intrinsic motivation and computational cost. (Osband et al., 2018)

It is noted that the training process of optimizing the value function of Bellman error has many similarities with the supervised learning. We use a stochastic gradient based Stochastic Gradient Langevin Dynamics (SGLD) to achieve posterior sampling (Welling & Teh, 2011). This work uses the Deep Deterministic Policy Gradient (DDPG) algorithm as the baseline algorithm (Lillicrap et al., 2015). We use the SGLD sampler instead of the value estimate function's optimizer to continuously generate a posteriori samples of the current observations during the training step, and use these samples to generate policy gradient optimization actors for exploration during the rollout phase. Further, we consider the parts of the original algorithm that are not coordinated with the posterior sampling and adjust them to obtain a more uniform sample. This is a more accurate way to sample from the posterior and introduces little additional computational cost.Further, we adjusted the portion of the baseline algorithm that is not coordinated with the posterior sampling to achieve a more consistent posteriori sampling. This is a more accurate way to sample from the posterior and introduces little additional computational cost.

In this work, we validated the performance of our exploration method with continuous MountainCar, Sparse HalfCheetah, and HalfCheetah as deceptive, sparse, and well-defined tasks, and compared it with Action noise, Parameter noise. In general, SGLD-based approaches can solve a wider variety of tasks with uniform settings and get better results in well-defined tasks. We summarize the three main contributions of this work:

- It is the first time to propose that the use of the SGLD method to achieve Thompson sampling in reinforcement learning algorithm which use neural network as Q-value function.
- Propose the methodology for using SGLD sampler to do Thomson sampling in off-policy actor-critic algorithm. No hyperparameter is introduced, and almost no additional computational overhead.
- This work proves through experiments that the method of this paper can solve the three different tasks of sparse reward, deception reward and good definition by using uniform settings.

This paper is organized as follows: In the section 2, we summarize some existing exploration methods and explain the relationship between our work and them. The third section briefly introduces the DDPG algorithm and the principle of SGLD, and gives a detailed description of how we apply SGLD in DDPG. In section 4 we conducted experiments on several representative tasks and analyzed why our methods are effective. Finally, we summarize this work in section 5 and point out our opinions on further research work.

2. Background

In this section, we first briefly introduce the actor-critic algorithm. Then compare well-defined tasks with sparse/deceptive tasks and explain why exploration strategies are the key to solving sparse/deceptive tasks. And finally, compare with the existing research work on exploration strategies.

We first define the basic symbols of reinforcement learning. The environment in the task of reinforcement learning is modeled as a Markov decision process: $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma >$, in which $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, in time step t, the agent observes state s_t and takes the action a_t, then recive reward $r_t \sim \mathcal{R}(s_t, a_t)$ and the state transfer to $s_{t+1} \sim \mathcal{P}(s_t, a_t)$. In an episode with time step $t \in 0 \cdots T$, the total discount return $G_T = \sum_{t=0}^{T} \gamma^t r_t$. For any given policy $\mu(s, a) = \mathbb{P}[a|s]$, start one episode from state s, $\mathbb{E}_\mu[G|s_0 = s]$ the expectation of total discount return can get from state s, also called state value function $V^\mu(s)$. Similarly, action state function $Q^\mu(s, a)$ can be defined as $\mathbb{E}_\mu[G|s_0 = s, a_0 = a]$.

2.1. Actor-Critic Algorithm

The actor-critic algorithm combines the advantages of the policy-based method and the value-based method to become one of the most popular reinforcement learning algorithms. This algorithm uses a parametric approach to represent the policy function or actor:$\pi(s, a|\theta^\pi)$ and the value function or critic $Q(s, a|\theta^Q)$, and solves the optimal parameters through optimization algorithms such as genetic algorithm, gradient descent, etc. In the actor-critic algorithm based on the gradient descent method, the policy function is optimized along the direction of policy gradient:$\nabla_{\theta^\pi} J = \mathbb{E}[\nabla_{\theta^\pi} \log \pi(s, a) \cdot Q(s, a)]$, and the value function is typically optimized by minimizing the mean square error $(R - V(s|\theta^V))^2$ with $R = \sum_{t=0}^{T-1} \gamma^t r_i + \gamma^T V(s_T)$ A general deep actor-critic algorithm framework is described in Algorithm 1.

It has been proven that, using advantage fuinction:$A(s, a) = Q(s, a) - V(s)$ instead of $Q(s, a)$ in policy gradient can reduce the variance and introduce no bias. Through the advantage function we can generate the intuition of how the critic affects the direction of policy gradient: In any state s^*, for any action a^* with $A(s^*, a^*) > 0$, the policy gradient will increase the probability of action a^* at state s^* $\pi(s^*, a^*|\theta^\pi)$. In other words, the agent will have a higher

probability of selecting the most advantageous action in the current state. This intuition will help us understand following part of this article about exploring in difficult environments.

Algorithm 1 General Deep Actor Critic Framework

Input: environment E, critic $V(s|\theta^V)$, actor $\pi(s|\theta^\pi)$, exploration strategy $\tilde{\pi} \leftarrow f(\pi)$
for 1 **to** cycle_number **do**
  Apply exploration strategy $\tilde{\pi} \leftarrow f(\pi)$
  Rollout data $\{d_t\}$ from E by $\tilde{\pi}$
  for 1 **to** update_number **do**
    Update Critic $V(s|\theta^V)$ by $\nabla_{\theta^V}(R - V(s|\theta^V))^2$
    Update Actor $\pi(s|\theta^\pi)$ by policy gradient
  end for
end for

2.2. Tasks With Different Reward Shape

Usually a reward for a task is defined according to the needs of the application. For example, in the HalfCheetah task, a half-body cheetah with 6 degrees of freedom is placed on the ground plane. In each time step, the reward is $r_t = r_{speed} + r_{control}$, where r_{speed} is proportional to the distance to the right in this time step, and $r_{control} = -\alpha * ||\vec{a_t}||_2^2$ with $\alpha = 0.1$ as default is the action penalty. The implication behind the reward is that the agent is expected to achieve higher speed with minimal control cost. Since the agent is constrained to a two-dimensional vertical plane, any random action can cause the agent to move to the right and immediately get a positive feedback. This means it is easy to sample a policy can get positive reward from policy space, and the positive reward will further drive the agent to learn how to run faster, thus forming positive feedback.

However, this well-defined situation does not always appear in all tasks, and positive feedback may not be easy to reach. In some tasks where the goal is to reach certain target states, a positive reward can only be obtained when the agent reaches these states. This means that the agent can only get 0 rewards before it happens to touch these states. The $Q(s, a)$ or $V(s)$ will learn to equal zero for any state and action, and cause the policy gradient be zero, the actor can not be optimized anymore. In addition, in the control task, in order to reduce the energy consumed by controlling each joint, include the action penalty term (as $r_{control}$ in HalfCheetah) will be included in the reward. Before getting any positive feedback, the agent will learn to keep the zero action as the best policy to minimize the penalty, this further prevents the agent from making various exploratory actions. In these cases, additional exploration strategies become especially important. Once the agent reaches the goal states and gains a positive reward through an additional exploration strategy, the critic will direct the actor learn to access the goal states with a higher frequency and finally learned to solve the task.

2.3. Exploration Strategies

The previous section briefly describes two factors that make the environment difficult to explore, sparse rewards and heavy penalty. The application requirement of "let the agent learn to achieve the goal with minimal cost" naturally leads to these two factors. In practical applications, due to insufficient understanding of the task or strict energy restrictions, it may lead to a task with sparse or deceptive nature. How to design an effective exploration strategy, so that agents can break through these difficulties, and quickly converge to the optimal policy becomes the key point to solve this kind of problem.

Some naive random search algorithms are generally used as default exploration strategies by various RL algorithms. Such as the ϵ-greed method which takes non-optimal action according to probability $1 - \epsilon$ (Mnih et al., 2015), or directly superimposes noise (correlation or not) in action space (Lillicrap et al., 2015). Another approach is to inject noise in the parameter space of the Q-value function of the policy function (Plappert et al., 2017). In this situation, the agent will take the same action in the face of the same state within an episode, which leads to more consistent exploration behavior. However, such methods do not explicitly suggest how to use existing data to guide exploration, resulting in low sample efficiency.

The Thompson sampling method, as an exploration strategy with a long history, has recently returned to the perspective of deep reinforcement learning researchers. This method draws a sample from the posterior distribution of the Q-value function and makes decisions based on the random sample rather than an optimal estimation. Since it is difficult to directly calculate the posterior distribution of the neural network, the key point in the work of using Thompson sampling in reinforcement learning is the approach to achieve posterior sampling. For example, BDQN (Osband et al., 2016) and prior-BDQN (Osband et al., 2018) uses a multi-head network trained in bootstrapping way and treats the set of output as sampled results. VIME (Houthooft et al., 2016) and Stein Variational Policy Gradient (Liu et al., 2017) use the reparameter trick and the stein gradient approach to implement variational inference as an approximation of the posterior distribution.

A serial "sanity checkes" have been proposed in prior-BDQN, include: 1.**Posterior Concentration**: requires the uncertainty of the posterior distribution of the value function decreases as the amount of observation data increases. 2.**Multi-step Uncertainty**: requires the uncertainty can be propagated through bellman equation. 3.**Epistemic vs**

Aleatoric: requires tfigure the posterior distribution of value function rather than the reward. 4.**Task-appropriate**: requires that the posterior distribution is on the parameter space of the q-function instead of the state space. 5.**Intrinsic motivation**: requires the value-function can give a non-zero value estimate of the unknown state with only zero reward recived. The last point is the key factor in whether the exploration strategy is effective in sparse reward tasks. Prior-BDQN, relative to BDQN, adds a random a priori function to the value function when optimizing the bellman error, which causes the intrinsic motivation. Coincidentally, in the work of parameter noise, adding random Gaussian noise directly to the Q-function or the actor can also lead to effective exploration, even without any clear considerations for bayesian learning. Combining these facts, we hypothesize that "noise on parameters" + "posterior sampling" may be important to the exploration strategy of deep reinforcement learning.

3. SGLD in Actor-Critic

Under the Bayesian learning framework, there is one mathematical method that has both the "posterior sampling" and "parameter noise" properties, called stochastic gradient Langevin dynamics. In this section, we will briefly introduce the SGLD and show how it can be used to construct an exploration strategy in Actor-Critic algorithms.

3.1. Stochastic Gradient Langevin Dynamic

Stochastic Gradient Langevin Dynamic algorithm is an algorithm that can perform MCMC sampling by stochastic gradient descent + Gaussian noise (Welling & Teh, 2011). For a paramter vector θ with a prior distribution $p(\theta)$,to draw a sample chain $\{\theta_1, \theta_2, \cdots\}$ follow $p(\theta|D)$ where $D = \{d_i\}_{i=1}^N$, the parameters should be updated as follow:

$$\Delta\theta = \frac{\epsilon}{2}\frac{N}{n}\sum_{i=1}^{n}\nabla_\theta \log p(d_i|\theta) + \frac{\epsilon}{2}\nabla_\theta \log p(\theta) + \mathcal{N}(0, \epsilon) \tag{1}$$

Where ϵ is the learning rate, n is the size of one mini-batch.

The basic SGLD algorithm updates all parameters with the same step size, which leads to slow mixing rate. In practical, we use preconditioned SGLD (pSGLD) instead, in which the step size is scaled by preconditioner G_t as follow:

$$\Delta\theta = \frac{\epsilon}{2}\frac{N}{n}G_t\sum_{i=1}^{n}\nabla_\theta \log p(d_i|\theta) + \frac{\epsilon}{2}G_t\nabla_\theta \log p(\theta) + \mathcal{N}(0, \epsilon G_t) \tag{2}$$

Where G_t is updated as the mean square term in RMSprop (Tieleman & Hinton, 2012).

To visualize the ability of pSGLD to characterization the uncertainty of posterior distribution when fitting data set, we trained a neural network with architecture of "20-ReLU-20-ReLU-1" by pSGLD on three toy datasets, left: $y = 0$, mid: $y = -x^2$, right: $y = x^3 - x$, and the results are shown in the figure 1. The dark curve is the average of the curve clusters, and the light areas represent the standard deviation of the curve clusters. The results show that the curve samples of pSGLD are concentrated near the data points, while diverge in both positive and negative directions in areas far from the data, even on the all zero data set.

3.2. Posterior sampling of critic

In Deep Actor Critic algorithm, the parameters of critic network θ^Q is updated by an SGD-like optimizer to minimize the MSE error $L^Q = \sum(R - V(S|\theta^Q))$. To replace the SGD-like optimizer by pSGLD sampler with minimal changes, we suppose that the likelihood term is $p(d_i|\theta^Q) \sim \exp(-L^Q)$ to match the Bellman error, and prior term is $p(\theta^Q) \sim \mathcal{N}(0, \sigma^2)$ to match the L2 regularization on critic network. Then the critic will updated as follow :

$$\Delta\theta^Q = -\frac{\epsilon}{2}\frac{N}{n}G_t\sum_{i=1}^{n}\nabla_\theta L^Q - \frac{\epsilon}{2}G_t\frac{\theta^Q}{\sigma^2} + \mathcal{N}(0, \epsilon G_t) \tag{3}$$

It is worth noting that, for off-policy algorithm with replay buffer, the size of data sets N in reinforcement learning is increasing as the training process increases. If the learning rate ϵ remains constant, the gradient of the likelihood term will continue to grow, resulting in an unstable training process. In order to keep the training process stable, we implement learning rate decay as opposed to data set size growth: $\epsilon_t = \frac{n}{N}\epsilon_0$. Now the equation (3) turns into:

$$\Delta\theta^Q = -\frac{\epsilon_0}{2}G_t\sum_{i=1}^{n}\nabla_\theta L^Q - \frac{\epsilon_0}{2}\frac{n}{N}G_t\frac{\theta^Q}{\sigma^2} + \mathcal{N}(0, \epsilon_0\frac{n}{N}G_t) \tag{4}$$

The observed data is increasing as the learning process progresses, which leads to the effect of the prior term and the noise term becomes weaker than the likelihood term. This is consistent with our common sense: the more data is observed, the more the model's distribution is concentrated toward the maximum likelihood estimate.

3.3. Exploration for Actor-Critic Algorithm

The Actor-Critic algorithm using SLGD is shown in algoritm 2. Critic $V(s|\theta^V)$ is continuously sampled by SGLD, so the last critic in the network is a sample follow $p(\theta^V|D)$ at any time. In the rollout phase, the current actor π is copied to $\tilde{\pi}$, then update $\tilde{\pi}$ use the policy gradient generated

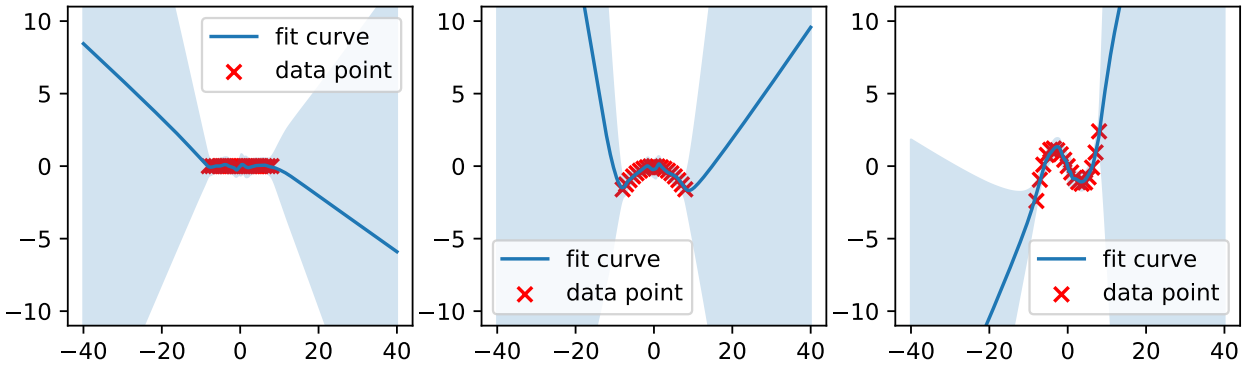


Figure 1. The results of curve fitting by using pSGLD on three toy data sets.

Algorithm 2 Deep Actor Critic with SGLD

Input: environment E, critic $V(s|\theta^V)$, actor $\pi(s|\theta^\pi)$, exploration strategy $\tilde{\pi} \leftarrow f(\pi)$
for 1 **to** cycle_number **do**
  Copy the actor for rollout $\tilde{\pi} \leftarrow \pi$
  for 1 **to** rollout_update_number **do**
    Update $\tilde{\pi}$ by policy gradient with last critic.
  end for
  Rollout data $\{d_t\}$ from E by $\tilde{\pi}$
  for 1 **to** update_number **do**
    Sample Critic $V(s|\theta^V) \sim p(\theta^V|D)$ with SGLD
    Update Actor $\pi(s|\theta^\pi)$ by policy gradient
  end for
end for

by the last critic. In Actor-Critic algorithm, the critic does not make decisions directly, so we optimize the copy of current actor with the last critic sample to get the optimal strategy of the last critic. We do this to follow the principle of Thompson sampling: making decisions based on the sampled value prediction model. The policy gradient with posterior sampled critic can estimated follow equation 5, which similar as (Henderson et al., 2017).

$$\nabla_{\theta^\pi}\mathbb{E}[L^\mu|D] = \mathbb{E}_{s_t\sim\rho,a_t\sim\pi,\theta^V\sim p(\theta^V|D)}[\nabla_{\theta^\pi}L^\pi] \quad (5)$$

Now we confirm the sanity check as (Osband et al., 2018) one by one: **Posterior Concentration**: As described in section 3.2, the intensity of the noise term and prior term will be weaker as more data is observed, and the distribution of critic will concentrate to the most likelihood estimation. **Multi-step Uncertainty**: The uncertainty of critic will be propagated throught the bootstrap value estimation target R. **Epistemic vs Aleatoric**: The SGLD sampler does posterior sampling follow $p(\theta^V|D)$ instead of $p(r_t|s_t, a_T)$. **Task-appropriate**: The SGLD sampler does posterior sampling in parameter space instead of state-action space which related to the environment. **Intrinsic motivation**: Even in zero reward environment, the SGLD sampler can give non-zero value estimation on unseen states. These analysis demonstrate that SGLD based exploration strategy can effectively overcome the shortcomings of existing exploration strategies.

4. Computational Experiment

In this section, we select MountainCar, SparseHalfCheetah and HalfCheetah three tasks as benchmarks to evaluate the method we proposed, representing deceptive rewards, sparse rewards, and well-defined tasks. The vanilla Deep Deterministic Policy Gradient algorithm was used as the no exploration baseline and implemented the action noise, parameter noise version as a comparison to our method. By visualizing the variance of the critic in the state-action space, we demonstrate that SGLD-based exploration strategy generate intrinsic motivation in sparse environments. The training results of the 4 methods on the three benchmarks are then given and compared. We used the same settings for all tasks and achieved the same results as the other methods, and some of the tasks exceeded. The evaluation results of the four methods on the three benchmarks are given and compared. We used uniform settings on all three tasks and solved all the tasks, surpassing other methods.

4.1. Experiment Setting

For each task, we run the training process for six times with different random seeds (which are same for different exploration strategies). For each training process, we train the agent with 1×10^6 transition data, and evaluate the agent with the learning policy(different from the rollout policy) for 10 times to get the average value of non-discount total return. More experiment setting is avaliable in Appendix A.

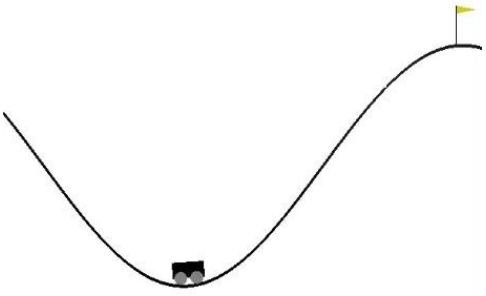
4.1.1. CMC,HC,SHC

We selected three continuous state-action space control tasks with different reward attributes as benchmarks, and all environments were modified from the original gym environments.

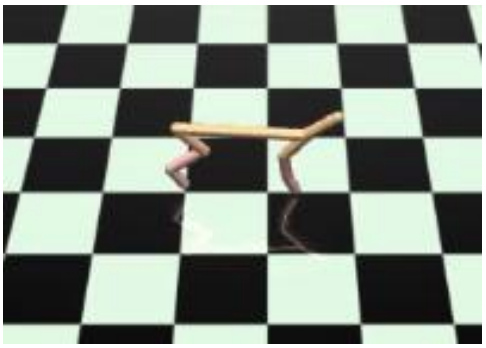
Continuous MountainCar (CMC): In the CMC, the agent needs to control a car from the bottom of the valley to the goal on the mountain top. The agent obtained reward $r = -\alpha * ||\vec{a_t}||_2^2 + r_{goal}$ in each time step and the goal reward $r_{goal} = 100$ only if the car reach the goal or zero otherwise. The car needs to climb the left side first and then rush to right to reach the top of the mountain, otherwise it can only oscillate back and forth in the valley. This is a task with both sparse rewards and action penalties.

HalfCheetah (HC): As described in section2.2, the agent needs to control a half cheetah to run as fast as possible to the right with minimal control. This is a well defined task with the default $\alpha = 0.1$, but as α increases, exploration becomes more and more difficult.

Sparse HalfCheetah (SHC): The SHC removes the action penalty from the HC and only receives a reward of 1 for each time step after the cheetah moves to the right beyond the distance d. This means that as d increases, it is more difficult for any random strategy to get a non-zero reward.



(a) Continuous MountainCar



(b) HalfCheetah

Figure 2. Snapshot of the gym environment.

4.1.2. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is an off-policy actor-critic algorithm that uses a replay buffer and two target networks (Lillicrap et al., 2015). In each training cycle, the agent rollouts some transition datas from the environment E and stores them into the replay buffer D. Then calculate the Bellman Error by equation (6) with the target network Q' and μ' and optimize the parameters θ^Q of the critic Q in several training steps. And optimize the parameters θ^μ of the actor μ by the policy gredient estimated by the critic. The target network is soft updated at the end of each training step. The details of the algorithm are described in Alg 3.

$$L^Q = \mathbb{E}_{s_t \sim \rho^{\tilde{\mu}}, a_t \sim \tilde{\mu}, r_t \sim E}[(Q(s_t, a_t) - y_t)^2] \quad \text{where} \quad y_t = \gamma Q'(s_{t+1}, \mu'(s_{t+1})) + r_t \tag{6}$$

Since DDPG is an off policy algorithm, the rollout actor and the learning actor do not need to be the same, the exploration strategy $f(\cdot) : \mathbb{M} \to \mathbb{M}$, which $\mathbb{M}$ is the space of policy, commonly transforms the learning actor μ into a rollout actor $\tilde{\mu}$ to achieve exploration.

Algorithm 3 Deep Deterministic Policy Gradient

Input: environment E, critic $Q(s, a|\theta^Q)$, actor $\mu(s|\theta^\mu)$,
exploration strategy $\tilde{\mu} \leftarrow f(\mu)$
Initialize replay buffer $D = \emptyset$.
Initialize target network $Q' = Q$, $\mu' = \mu$.
for 1 **to** cycle_number **do**
  Apply exploration strategy $\tilde{\mu} \leftarrow f(\mu)$
  Rollout data d_t from E by $\tilde{\mu}$ and $D \leftarrow D \cup d_t$
  for 1 **to** train_steps **do**
    Sample data batch $\{d_t = (s_t, a_t, r_t)\}$ from D
    $L^Q = \sum(\gamma Q'(s_{t+1}, \mu'(s_{t+1})) + r_t - Q(s_t, a_t))^2$
    $L^\mu = -\sum Q(\mu(s_t))$
    $\theta^Q \leftarrow \theta^Q - \alpha^Q \cdot \nabla_{\theta^Q} L^Q$
    $\theta^\mu \leftarrow \theta^\mu - \alpha^\mu \cdot \nabla_{\theta^\mu} L^\mu$
    Soft update target network Q' and μ'
  end for
end for

In original DDPG algorithm, the target networks are the τ-moving average of parameters of actor and critic. After each gradient descent step on actor and critic ,the target networks are "soft updated" by equation (7).

$$\theta^{Q'} = (1 - \tau)\theta^{Q'} + \tau\theta^Q \qquad \theta^{\mu'} = (1 - \tau)\theta^{\mu'} + \tau\theta^\mu \tag{7}$$

The optimized object for the critic in one cycle is minimizing Bellman error in equation (6), which is associated with both data in replay buffer and the target network. Constantly updating the target during the training step will make the

process of sampling from the posterior distribution inconsistently. In order to sample critic from posterior distribution more strictly, we remain the target unchanged in one cycle and do "hard update",directly copy parameters of actor and critic to target networks. Under this setting, the expectation policy gradient will be intergrated over both $s_t \sim \rho$ and $\theta^Q \sim p(\theta^Q|D)$ as equation (8).

$$\nabla_{\theta^\mu}\mathbb{E}[L^\mu|D] = \mathbb{E}_{s_t\sim\rho,\theta^Q\sim p(\theta^Q|D)}[\nabla_{\theta^\mu}L^\mu] \tag{8}$$

It is proven to be the correct form to estimate policy gradient with critics sampled from posterior distribution in (Henderson et al., 2017).

4.2. Intrinsic Motivation in State-Action space

In the MC task, the state space is a two-dimensional space in which the position and velocity are formed. To show how SGLD is intrinsically motivated, we visualize the uncertainty of Q(s,a). The agent fixes a=1 and rollsout the data, the car will reciprocate, and the observed data is a circle in the phase space. We then use this data to train the Adam version and the SGLD version of the DDPG, and record 50 critic samples, collect the surfaces of these samples on the State plane, and count the variance at each point, as shown in figure 3.

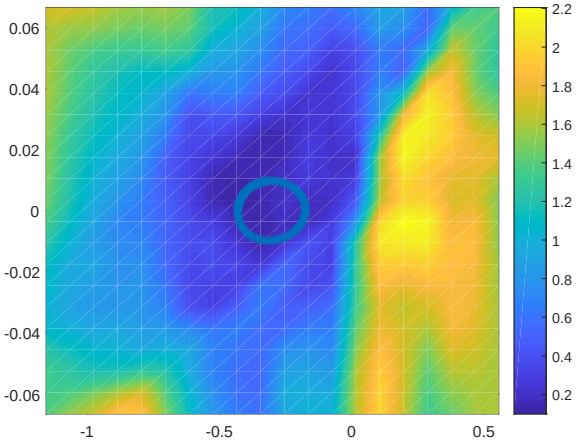


Figure 3. The variance plot of critics sampled by SGLD on Continuous MountainCar

As can be seen from the figure, the Adam version of the critic converges to a unique output, and the difference above the entire plane is almost 0. The slash version of the SGLD has a smaller variance near the data circle, and in the far away area. Large variance. This shows that using SGLD can indeed produce a posteriori sampling of critic.

4.3. Exploration and Exploitation

We evaluate the performance of our method on MC, HC and SHC and compared it with the results of no exploration baseline, action noise and parameter noise.

On the MC, including no action penalty, no exploration and parameter noise all failed. The action noise and SGLD methods succeeded with high probability under all conditions. It can be observed that as the action penalty increases, the probability of the SGLD solving the task decreases, and the action noise is almost unaffected. This is because the OU process can almost always ignore the subject strategy to achieve the goal in the case of std=0.6, which leads to unconditional target reward, but this success is difficult to extend to other tasks.

Table 1. Number of Succeeded Seed on Continuous MountainCar

α	Baseline	SGLD	AN	PN
0	0	6	6	0
0.1	0	6	5	0
0.2	0	5	6	0
0.5	0	4	6	0
0.1	0	3	6	0

On the HC, as the action penalty increases, the task becomes difficult to explore. Under the settings of alpha=0, 0.1 and 0.2, all algorithms showed a consistent learning curve despite different learning speeds. It can be assumed that under this setup all algorithms work. When alpha=0.5 and 1, the no exploration and action noise methods almost fail, and the parameter noise SGLD method can still maintain the same learning curve as before. We think this is because the rollout policy of the two exploration methods gives the same action in the same state, and the action noise can not give a consistent action, and the no exploration is difficult to explore due to the action penalty.

Table 2. Mean Return on HalfCheetah

α	Baseline	SGLD	AN	PN
0	3843	3742	3454	3776
0.1	2813	4091	1683	2977
0.2	2352	3678	931	2801
0.5	214	2464	3	1641
1	481	984	684	1605

On the SHC, at the distance d=1, the three exploration strategies successfully solved the problem, Even the no exploration baseline has one seed reached non-zero rewards. As the interval d increases, the probability that baseline and action noise can explore non-zero rewards is getting lower

and lower. But SGLD and parameter noise are stable to solve the problem, which further confirms the conclusion about deep exploration that we obtained on HC.

Table 3. Number of Succeeded Seed on Sparse HalfCheetah

d	Baseline	SGLD	AN	PN
1	1	6	6	6
2.5	0	5	1	6
5	0	6	0	5
10	0	4	0	5
20	0	4	0	2

In conclusion: The Action Noise method succeeded on MC, but was unable to solve difficult($d \geq 5$) SHC tasks. The Parameter Noise method succeeded in the SHC task, but failed on the MC task. The baseline almost failed in all tasks. Our method solved all tasks with a high probability of all settings (minimum 3/6, average 4.9/6). All methods, including the baseline method, can learn to run continuously on the HC with light action penalty. However, as the action penalty increases, the total return attenuation of our method and parameter noise method is slower than the baseline and Action noise. The total return of our method and parameter noise method is less reduced, while significant reductions were observed in experiments of baseline and action noise. This shows that our approach has the best generalization ability on all tasks, and can solve difficult problems and well-defined problems at the same time.

4.4. Application Friendly

It is worth noting that our method does not introduce any hyperparameters. In the experiments shown above, we use the same settings in all tasks, and achieved success in both well-defined tasks and hard tasks. Which means lower cost of tuning hyperparameters. Our approach also introduces little additional computational cost. We ran our method on the MC and compared it to the baseline method. The results show that our method has a close running time (XX: XX) compared to the baseline, which proves that our method is computational efficient.

5. Conclusion and Further work

This article summarizes the reasons that lead to difficulties in exploring sparse/depetive tasks as "zero reward" and "action penalty" and indicates how these factors affect reinforcement learning algorithms. We propose an exploring strategies using SGLD to achieve Tompson sampling in Actor-critic algorithm, which overcomes the shortcomings of existing algorithms and achieve effectively explore in sparse/deceptive tasks, with no additional hyperparameters are introduced and is computationally friendly. We use the same settings to evaluate our method on three continous tasks, and the results show that our method can solve more tasks and the sample is more efficient.

Our work demonstrates the exploration strategy with combination of parameter noise and posterior sampling in reinforcement learning. This algorithm continuously generates the sampling of the value function under all observed data and the corresponding policy during the training process. This approach may be combined with evolution algorithms as a way to generate new populations, which may cause cover the policy parameter space better and make full use of the data for each time step. In addition, the intensity of exploration in our approach is related to the number of data observed, which means throwing away some of the redundant data will "restart" the exploration, combined with further data management strategy design may lead to better exploration.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Azizzadenesheli, K., Brunskill, E., and Anandkumar, A. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

Bakker, B. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, pp. 1475–1482, 2002.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K., and Clune, J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, pp. 5032–5043, 2018a.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K., and Clune, J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, pp. 5032–5043, 2018b.

Henderson, P., Doan, T., Islam, R., and Meger, D. Bayesian policy gradients via alpha divergence dropout inference. *arXiv preprint arXiv:1712.02037*, 2017.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

Khadka, S. and Tumer, K. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1196–1208, 2018.

Lehman, J. and Stanley, K. O. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., and Wierstra, D. Continuous control with deep reinforcement learning. *Computer Science*, 8(6):A187, 2015.

Liu, Y., Ramachandran, P., Liu, Q., and Peng, J. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., and Ostrovski, G. J. N. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015. ISSN 1476-4687.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

Osband, I., Aslanides, J., and Cassirer, A. Randomized prior functions for deep reinforcement learning. *arXiv preprint arXiv:1806.03335*, 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Riquelme, C., Tucker, G., and Snoek, J. Deep bayesian bandits showdown. In *International Conference on Learning Representations*, 2018.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z., et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., and Lanctot, M. J. n. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016. ISSN 1476-4687.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2753–2762, 2017.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

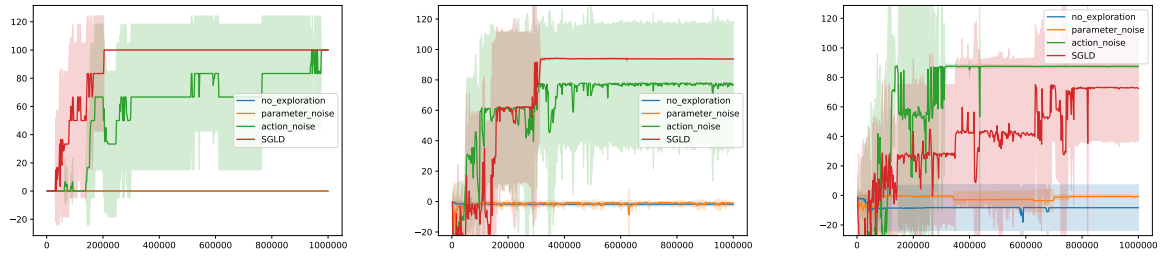
Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 681–688, 2011.

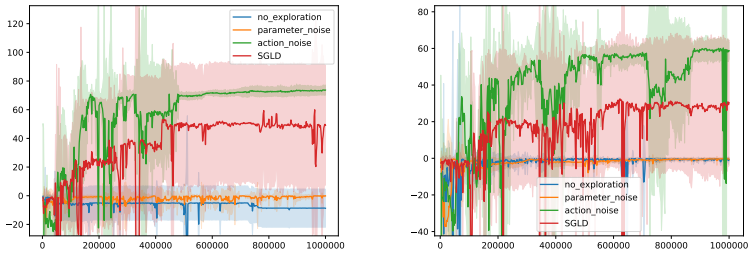
A. Details of experiment setting

Our test machine is equipped with an Intel i9-7920X CPU and two GTX1080Ti GPU. The operating system is Ubuntu 16.04 LTS and uses the configuration of python3.5.2 + torch0.4.1 + gym0.10.8 + mujoco-py1.50.1.62 + mujoco150.

B. Figures for training process

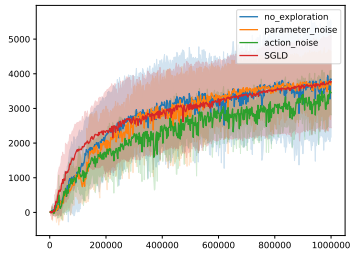


(a) $\alpha = 0$ (b) $\alpha = 0.1$ (c) $\alpha = 0.2$

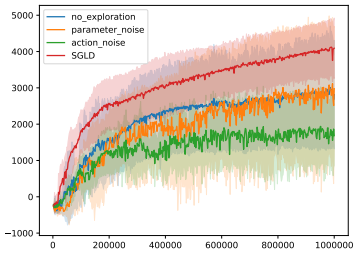


(d) $\alpha = 0.5$ (e) $\alpha = 1$

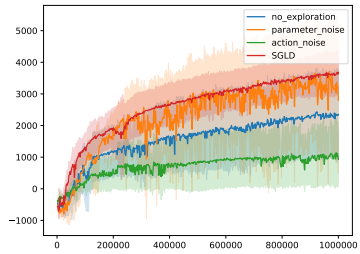
Figure 4. MountainCar



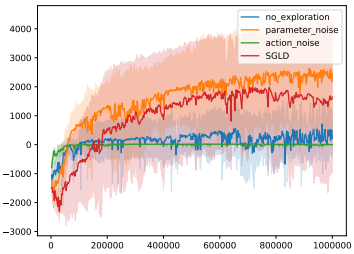
(a) $\alpha = 0$



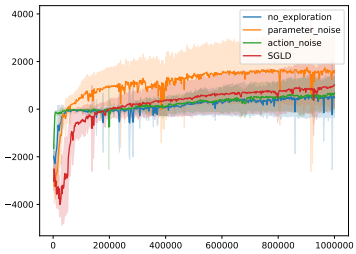
(b) $\alpha = 0.1$



(c) $\alpha = 0.2$

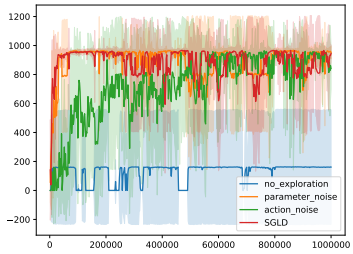


(d) $\alpha = 0.5$

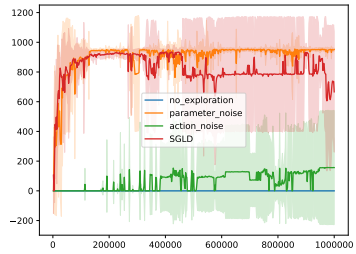


(e) $\alpha = 1$

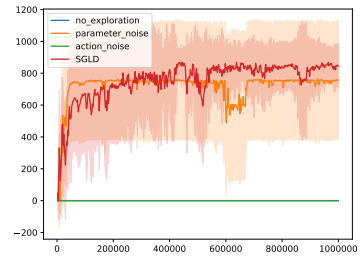
Figure 5. HalfCheetah



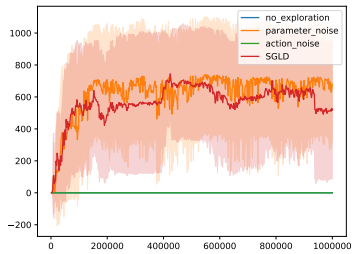
(a) $d = 1$



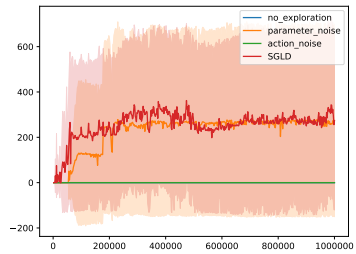
(b) $d = 2.5$



(c) $d = 5$



(d) $d = 10$



(e) $d = 20$

Figure 6. Sparse HalfCheetah