Reproducibility report of ”Deterministic Policy Gradient Algorithms”

Yueliang Xie
SUN YAT-SEN UNIVERSITY
Email:1125666900@qq.com

Abstract. This report is mainly a brief introduction and review of the article ”Deterministic Policy Gradient Algorithms”, as well as the test results of a simple code that is reproduced and implemented based on this article. The article is a must-read article of the key papers on OpenAI Spinning up, so it is also very milestone.

Keywords: DPG, Deepmind, etc.

1 Introduction

This article is the predecessor of the DDPG algorithm, and is also a required article to understand the Deterministic Policy Gradient (DPG) algorithm. I hope to understand what DPG algorithm is and why it works by reading this article.

The authors introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. The authors demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

Policy gradient algorithms are widely used in reinforcement learning problems with continuous action spaces. The basic idea is to represent the policy by a parametric probability distribution$\pi_\theta(a \mid s) = P[a \mid s; \theta]$ that stochastically selects action a in state s according to parameter vector θ. Policy gradient algorithms typically proceed by sampling this stochastic policy and adjusting the policy parameters in the direction of greater cumulative reward.

In this paper the authors instead consider deterministic policies a = $\mu_\theta(s)$.From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.

In addition, the author also proves that DPG is a special case of SPG, that is, when the strategy variance of SPG tends to 0, SPG is DPG.

In the stochastic policy gradient SPG, we need to integrate the state and action at the same time:

$$
\begin{aligned}
\nabla_\theta \mathbf{J}(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s,a) da ds \\
&= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \log \pi_\theta(a \mid s) \pi_\theta(a \mid s) \\
&\quad Q^\pi(s,a) \mathrm{d}a \ \mathrm{d}s \\
&= E_{s \sim \rho^\pi, a \sim \pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)\right]
\end{aligned}
$$

Then it is necessary to sample the state and action under the corresponding distribution. When the dimension is high, the sampling amount will increase. In other words, SPG needs to have enough sampling amount to correctly output the correct strategy distribution to select the correct one. Action, once the action dimension is high, the workload is still very large.

However, the DPG introduced by the author only needs to integrate the state, and the integration of the action is less, that is, there is no need to sample a large number of actions, which improves the efficiency.

2 Methodology

Since the original paper's code is not open source, with very less descriptions and guidance, I read through the whole article and comprehend the functions and implementation methods of each part. After consulting the literature, I found that TensorFlow is the most commonly used implementation on the network, and I am familiar with pytorch. So I implemented the code with pytorch by referring to the version of TensorFlow and ran it successfully.

3 Background

We study reinforcement learning and control problems in which an agent acts in a stochastic environment by sequentially choosing actions over a sequence of time steps, in order to maximise a cumulative reward. We model the problem as a Markov decision process (MDP) which comprises: a state space $\mathcal{S}$, an action space $\mathcal{A}$, an initial state distribution with density $p_1(s_1)$, a stationary transition dynamics distribution with conditional density $p(s_{t+1} \mid s_t, a_t)$ satisfying the Markov property $p(s_{t+1} \mid s_1, a_1, \ldots, s_t, a_t) = p(s_{t+1} \mid s_t, a_t)$, for any trajectory $s_1, a_1, s_2, a_2, \ldots, s_T, a_T$ in state-action space, and a reward function r: $\mathcal{S} \times \mathcal{A} \rightarrow R$. A policy is used to select actions in the MDP. In general the policy is stochastic and denoted by $\pi_\theta : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the set of probability measures on $\mathcal{A}$ and $\theta \in R^n$ is a vector of n parameters, and $\pi_\theta(a_t \mid s_t)$ is the conditional probability density at $a_t$associated with the policy. The agent uses its policy to interact with the MDP to give a trajectory of states, actions and rewards, $h_{1:T} = s_1, a_1, r_1 \ldots, s_T, a_T, r_T over \mathcal{S} \times \mathcal{A} \times R$. The return r_t^γ is the total discounted reward from time-step t onwards, $r_t^\gamma = \sum_{k=t}^\infty \gamma^{k-t} r(s_k, a_k)$ where $0 < \gamma < 1$. Value functions are defined to be the expected total discounted reward, $V^\pi(s) = E\left[r_1^\gamma \mid S_1 = s; \pi\right]$ and $Q^\pi(s,a) = E\left[r_1^\gamma \mid S_1 = s, A_1 = a; \pi\right]$

. The agent's goal is to obtain a policy which maximises the cumulative discounted reward from the start state, denoted by the performance objective $J(\pi)=E\left[r_1^\gamma \mid \pi\right]$

3.1 Stochastic Policy Gradient Theorem

The most basic idea of SPG is that since my goal is to maximize the objective function, then directly adjust the policy parameters to the direction of the gradient of the objective function (the direction of the gradient is the direction of the large rate of change), that is, the gradient rises, and the adjusted The parameters of the strategy network, then it is equivalent to get the optimal strategy. The fundamental result underlying these algorithms is the policy gradient theorem:

$$\begin{aligned}\nabla_\theta J\left(\pi_\theta\right) & =\int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s, a) \mathrm{d} a \mathrm{~d} s \\ & =E_{s \sim \rho^\pi, a \sim \pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a)\right]\end{aligned} \tag{1}$$

The superscript π of Q indicates that the incoming Q value comes from the Q obtained by the target strategy in value-based).

The first to second steps are derived from the definition of expectation. Both s and a are random variables. The expectation is determined by the distribution of R. V. Which distribution X belongs to is the mathematical expectation of this distribution. In addition, it can be regarded as the probability density, that is, the distribution of R. V.

3.2 Stochastic Actor-Critic Algorithms

The actor-critic is a widely used architecture based on the policy gradient theorem (Sutton et al., 1999; Peters et al., 2005; Bhatnagar et al., 2007; Degris et al., 2012a). The actor-critic consists of two eponymous components. An a c tor adjusts the parameters θ of the stochastic policy $\pi_\theta(s)$ by stochastic gradient ascent of Equation 1. Instead of the unknown true action-value function $Q^\pi(s, a)$ in Equation 1 , an action-value function $Q^w(s, a)$ is used, with parameter vector w . A critic estimates the action-value function $Q^w(s, a) \approx Q^\pi(s, a)$ using an appropriate policy evaluation algorithm such as temporal-difference learning. In general, substituting a function approximator $Q^w(s, a)$ for the true action-value function $Q^\pi(s, a)$ may introduce bias. However, if the function approximator is compatible such that i) $Q^w(s, a)=\nabla_\theta \log \pi_\theta(a \mid s)^{\top} w$ and ii) the parameters w are chosen to minimise the mean-squared error $\epsilon^2(w)=E_{s \sim \rho^\pi}$, a $\sim \pi_\theta\left[\left(Q^w(s, a)-Q^\pi(s, a)\right)^2\right]$, then there is no bias (Sutton et al., 1999),

$$\nabla_\theta J\left(\pi_\theta\right)=E_{s \sim \rho^\pi, a \sim \pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^w(s, a)\right] \tag{2}$$

3.3 Off-Policy Actor-Critic

The off-policy algorithm consists of two different strategies, and the typical off-policy is the Q-learning algorithm. One is a behavior strategy and the other is a targeted strategy. It is often useful to estimate the policy gradient off-policy from trajectories sampled from a distinct behavior policy. Differentiating the performance objective and applying an approximation gives the off-policy policy gradient.

4 Gradients of Deterministic Policies

Now considering how the policy gradient framework may be extended to deterministic policies. The main result is the deterministic policy gradient theorem, similar to the stochastic policy gradient theorem introduced in the previous section. The author gives informal and formal derivation methods in sections 3.1 and 3.2 of the original text, respectively.

4.1 Action-Value Gradients

Policy evaluation is responsible for predicting value functions in RL tasks, usually using DP, MC or TD, and policy improvement usually uses greedy strategies.Policy evaluation methods estimate the action-value function, for example by Monte-Carlo evaluation or temporal-difference learning. However, this strategy improvement method can still be used in discrete action spaces. If it is in continuous action spaces, then you have to find the global maximum of the value function, which is obviously impractical. Therefore, the author proposes a new way of thinking: Adjust the policy parameter towards the direction of the gradient of the value function Q. For each traversed state s, the parameter will be updated. Considering that the degree of single-step update of the state is different, small batch updates will be used to improve the stability of the update.

4.2 Deterministic Policy Gradient Theorem

Give the deterministic policy gradient theorem:

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &= \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)\Big|_{a=\mu_\theta(s)} \mathrm{d}s \\ &= E_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)\big|_{a=\mu_\theta(s)}\right]\end{aligned} \tag{3}$$

From this formula, we can see that the gradient only integrates the state. Compared with the random strategy, the integration space for the action is greatly reduced, which means that there is no need to sample the action, which greatly improves efficiency.

4.3 Limit of the Stochastic Policy Gradient

The authors show that deterministic policy gradients are the limiting case of stochastic policy gradients when the policy variance tends to zero.

$$\lim_{\sigma\downarrow 0} \nabla_\theta J(\pi_{\mu_\theta,\sigma}) = \nabla_\theta J(\mu_\theta) \tag{4}$$

This is an important result because it shows that familiar policy gradient mechanisms, e.g., are compatible with function approximation (Sutton et al., 1999), natural gradients (Kakade, 2001), actor-critic (Bhatnagar et al., 2007) or episodic/batch methods (Peters et al., 2005), also suitable for deterministic policies gradient.

5 Deterministic Actor-Critic Algorithms

Deterministic policies can be used in both on-policy and off-policy algorithms. We can use Sarsa as the critic for the on-policy algorithm, and for the off-policy algorithm, we can use Q-learning as the critic. The author also proposed two conditions similar to the elimination of bias in the randomness strategy.

5.1 On-Policy Deterministic Actor-Critic

A major disadvantage of the deterministic strategy is that the output is a deterministic value, which lacks exploratory nature and is easy to fall into a local optimum. Therefore, a little noise (OrnsteinUhlenbeck noise, Gaussian noise, etc.) can be artificially added to the output to reflect the exploratory nature.

On-policy Deterministic Actor-Critic consists of 2 parts: one part is the on-policy critic, such as Sarsa is used to predict the Q value, and the part of the policy improvement is that the Actor continuously improves the Q value. For s1, it can output a deterministic a1; the other part is the Actor-network, and the improvement of the parameters of the deterministic Actor network is no longer in the form of multiplying the policy gradient and the critic, but in the direction of the gradient of the Q value. Like the randomness strategy, we still use a value function approximator instead of the true value of the Q.

5.2 Off-Policy Deterministic Actor-Critic

We now consider off-policy methods that learn a deterministic target policy $\mu_\theta(s)$ from trajectories generated by an arbitrary stochastic behavior policy$\pi(s,a)$. As before, we modify the performance objective to be the value function of the target policy, averaged over the state distribution of the behavior policy.

The algorithm consists of two parts, one part is Critic, which uses a Q-learning algorithm to approximate the true value of Q: The other part is the Actor, which is the same as the on-policy Deterministic AC algorithm in Section 4.1 of the original text. The update of the parameters follows the direction of the gradient of the Q value.

Since Critic uses Q-learning, Actor uses a deterministic strategy, which does not integrate or sample actions. Therefore, both of them do not require importance sampling and do not require IS correction factors.

5.3 Compatible Function Approximation

Continuing the two conditions of the random strategy, the author also proposed two conditions for the deterministic strategy:
1. $\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s)^\top w \quad and$
2. w minimizes the mean-squared error, $\text{MSE}(\theta,w) = E\left[\epsilon(s;\theta,w)^\top \epsilon(s;\theta,w)\right]$ where $\epsilon(s;\theta,w) = \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$

Finally, we show that the natural policy gradient (Kakade, 2001; Peters et al., 2005) can be extended to deterministic strategies.

6 Opinion

Three major advantages of DPG:

(1): Adjust the strategy parameter towards the direction where the gradient of the value function Q increases.

(2): The performance objective of the deterministic strategy gradient only integrates the state. Compared with the random strategy, the integration space for the action is greatly reduced, which means that there is no need to sample the action, which greatly improves efficiency.

(3): The actor uses a deterministic strategy, which does not integrate or sample actions, and does not require an IS correction factor. DPG, which is a deep variant of the deterministic policy gradient algorithm, uses a Q-function estimator to enable off-policy learning and a deterministic actor that maximizes this Q-function.

As such, this method can be viewed both as a deterministic actor-critic algorithm and an approximate Q-learning algorithm. Unfortunately, the interplay between the deterministic actor-network and the Q-function typically makes DDPG extremely difficult to stabilize and brittle to hyperparameter settings. As a consequence, it is difficult to extend DPG to complex, high-dimensional tasks, and on-policy policy gradient methods still tend to produce the best results in such settings.

7 Experimental setup and code

CPU	AMD EPYC 7742 64-Core Processor
RAM	252G
GPU	NVIDIA A100 40G
OS	Debian 11
Compiler	GNU@7.5.0
CUDA	CUDA@11.3.1
Pytorch	Torch@1.13.0+cu117

The original text does not give the specific code of the implementation, so I can only restore the implementation of DPG and SAC according to the ideas given in the original text. Due to time reasons, I conducted experiments under the game environment CartPole provided by the gym that comes with python and made differences by comparing DPG and SAC. Due to reasons such as experimental equipment and time conditions, I did not conduct relevant tests on other games involved in the paper. Since the test environment, data sets, and other environmental factors are different from those in the paper, and the code is written by myself, there may be a big difference in implementation, so the final result may be slightly different. However, the results obtained on the big topic are consistent with the original text, that is, the introduction of DPG has greatly improved compared with the original SAC, and there are significant advantages in both training speed and final results.

8 Result show

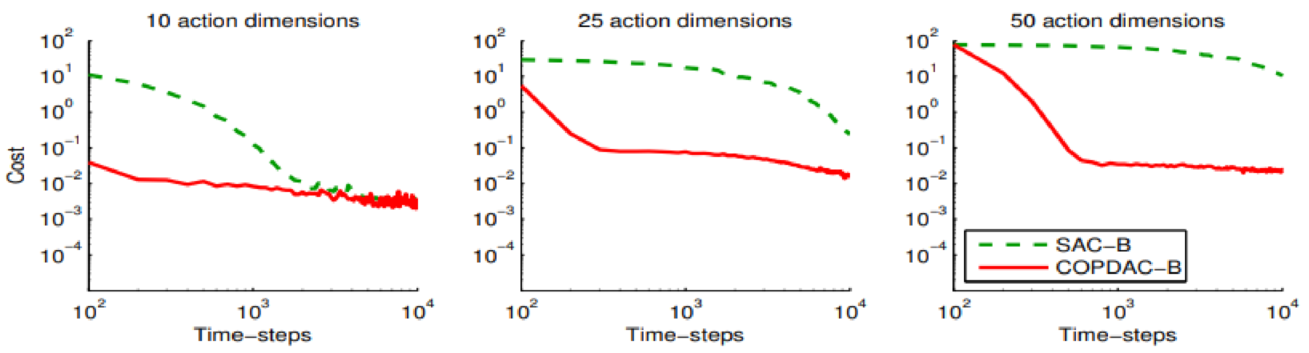


Figure 1. Comparison of stochastic actor-critic (SAC-B) and deterministic actor-critic (COPDAC-B) on the continuous bandit task.

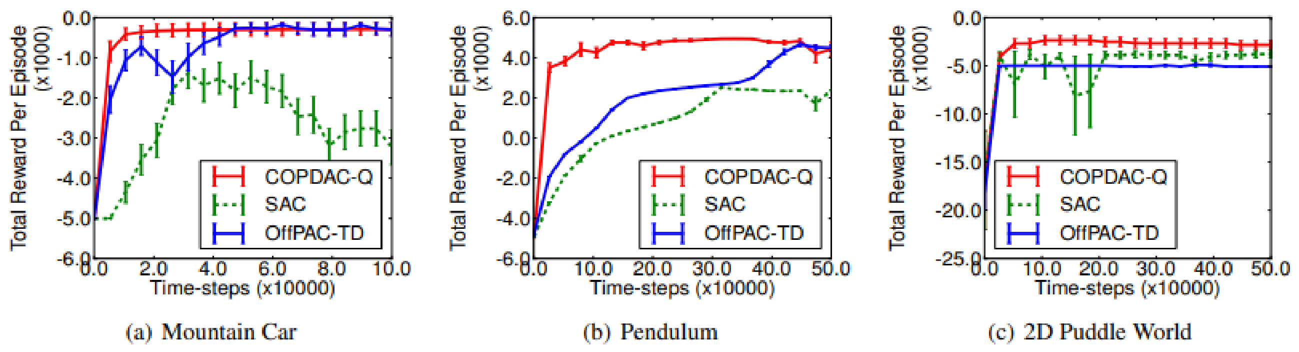
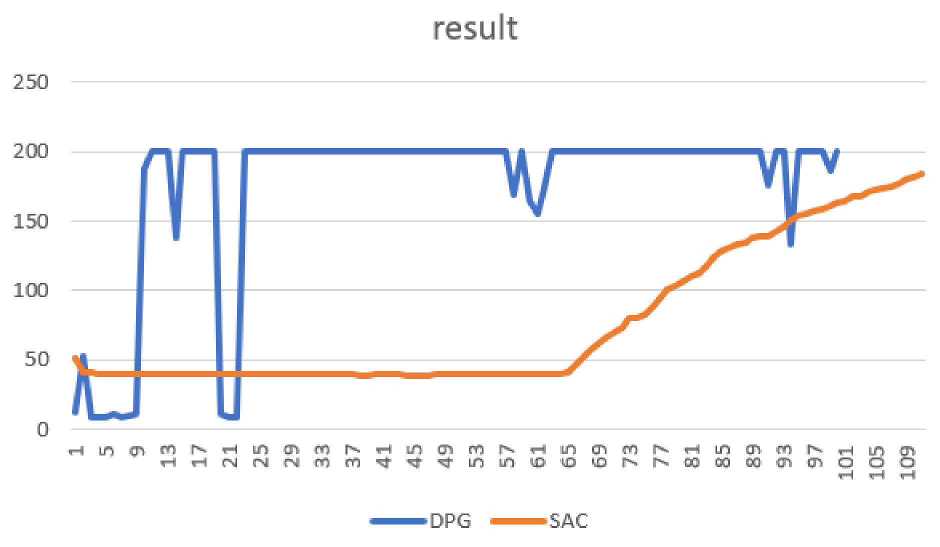
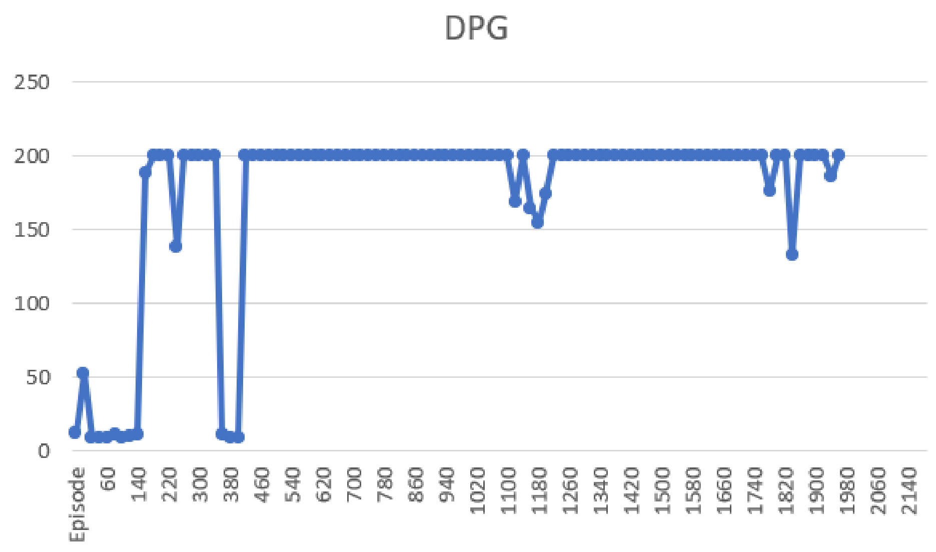


Figure 2. Comparison of stochastic on-policy actor-critic (SAC), stochastic off-policy actor-critic (OffPAC), and deterministic off-policy actor-critic (COPDAC) on continuous-action reinforcement learning. Each point is the average test performance of the mean policy.

After running, the output of the code are as followed. Above is the result of the original paper.Then I compare them with the result shown in the article.





The code is open source by the link:(https://github.com/Moon1125666900/RLfinal). And the file *SAC.py* and the *DPG.py* are implementation of SAC and DPG respectively.

9 Summary

This time I tried to re-implement the DPG algorithm framework according to the ideas in the paper Deterministic Policy Gradient Algorithms. Since the original code is not open source, it is relatively old, and there is no related implementation on the Internet, so I encountered many difficulties in the process of implementing the code. However, after consulting relevant literature and carefully analyzing and understanding the ideas of the original text, I finally realized a relatively basic version. However, the games involved in the original text are many and complicated, so it is a pity that there is not enough time to reproduce all the games involved in the article.

Through this experiment, my understanding of reinforcement learning has also been further improved. There are more mathematical formula derivations in the original text, and I feel that I have learned a lot during the derivation process. At the same time, my coding ability has also been further improved in this experiment. Knowing about their principle and implementation methods, as well as their convergence and efficiency in application to the problems. I believe this will also be of great help to future learning.

10 Reference

1. Deterministic Policy Gradient Algorithms. [1]

2. Mofan's machine-learning [2]

References

[1] David Silver, G. L. and Nicolas Heess Thomas Degris, D. W. M. R. “Deterministic Policy Gradient Algorithms”. In: *International Conference on Machine Learning* 32 (2014).

[2] “https://mofanpy.com/tutorials/machine-learning/reinforcement-learning”. In: ().