Off-Policy Algorithms for Continuous Control

Khoa Tran, , Luc Truong, and Vu Huynh
University of Information Technology, Ho Chi Minh City, Vietnam
Vietnam National University, Ho Chi Minh City, Vietnam
{20520222, 20520241, 20520864}@gm.uit.edu.vn,

***Abstract*—In this paper, we will present some off-policy policy gradient algorithms. the goal of these algorithms is to find an optimal policy for continuous control tasks. Using the actor-critic framework and Policy Gradient, there are three algorithms that will be present in this paper: Deep Deterministic Policy Gradient, Twin Delayed DDPG, and Soft Actor-Critic. We will write about how these algorithms work and the performance of these algorithms on 3 environments with 4 random seeds. To confirm the enhanced performance of these algorithms we will give the source code together with the paper. There is also a video about these algorithms that come with the paper.**

***Index Terms*—Reinforcement learning, Neural Network, Deep Reinforcement Learning**

I. Introduction

Reinforcement learning combined with deep learning has proven its robustness through solving complex control problems. Deep Q-Learning (DQN) algorithm succeeds in training an agent to plays atari game agent from pixel images. However, this algorithm still has many challenges, one of which is that this algorithm can only solve tasks with discrete action space.

The off-policy algorithms we describe DDPG, SAC and TD3 demonstrate their performance by being able to solve high-dimensional tasks in both action and state space. As well as learn the methods these algorithms use to solve problems like overestimate bias, exploration, and more. In each algorithm, we will talk about the main idea, the limit of the previous methods and how this method can solve that problem.

II. Related Work

Many other approaches can also solve reinforcement learning problems. First, on-policy algorithms, unlike off-policy ones, try to improve the policy which choose the action to interact with the environment, behavior policy and target policy are the same. Second, evolution strategy (ES), a group of search/optimization algorithms inspired by nature. Thirdly, cleverly combining ES and RL algorithms for impressive performance, such as CEM-TD3 and CEM-SAC.

III. Background

A. Reinforcement Learning

In reinforcement learning, the learner and decision-maker is called the agent. This agent learns from the experiences it receives when it interacts with the environment. This is often modeled as a Markov decision process (MDP). At each timestep t, agent in state $s_t \in \mathcal{S}$, perform an action $a_t \in \mathcal{A}$ following a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, receives a reward r_t and transitions to the next state s_{t+1}. The goal of RL is to find an optimal policy-π^* = $\underset{\pi}{argmax}\ \mathcal{J}(\pi)$:

$$\mathcal{J}(\pi) = \mathbb{E}_{\pi}[\sum_{t\geq 0} \gamma^t r_t] \tag{1}$$

where $\mathcal{J}(\pi)$ is the cumulative discounted reward with discount factor $\gamma \in [0, 1]$.

The policy π can be parameterized by the parameter θ. In Deep Reinforcement Learning (DRL), θ would be the weight vector representing a deep neural network π_θ. Then training the task to find the optimal policy is equivalent to finding a set of weights θ^* to get the maximum reward function:

$$\theta^* = \underset{\theta}{argmax}\ \mathcal{J}(\pi_\theta) \tag{2}$$

Next, we will describe about policy gradient, a popular families of policy search algorithms.

B. Policy Gradient

A group of methods commonly used in DRL algorithms is called the Policy Gradient. The main idea of this method is to update the parameter θ based on the gradient of J(π_θ) with respect to the policy parameter[1].

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathcal{J}(\pi_{\theta_t}) \tag{3}$$

where $\mathcal{J}(\pi_\theta)$ can be computed by using *policy gradient theorem*:

$$\nabla_\theta \mathcal{J}_{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)] \tag{4}$$

where $Q^{\pi_\theta}(s, a)$ is the state-action value function, show that how much cumulative discounted reward the agent can get if it in state s perform a action a and thereafter following policy π_θ .

Actor-Critic is an architecture based on a policy gradient theorem. Actor-Critic has two main components. A *critic* used to approximate the Q-function, showing how well the action is chosen by the actor. Critic update parameters **w** by minimizing the loss function:

$$\mathcal{J}(\mathbf{w}) = r_t + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a) \tag{5}$$

An *actor* update policy parameters in the direction suggested by the critic using the formula (Equation 3). The actor represents the agent's policy, telling the agent what action to take in each state. The objective function of the actor is the sum of the cumulative discounted rewards (Equation 1).

IV. METHODS

A. Deep Deterministic Policy Gradient (DDPG)

DQN is a Q learning algorithm that uses a neural network as a function approximator to solve tasks with large state spaces. However, DQN only works in an environment with a discrete action space, this is because in a task with a continuous action space, it is not possible to find a greedy policy (Equation 2) at each timestep. To solve this problem, the author of DDPG algorithm uses an actor-critic architecture based on the DPG algorithm.

The DPG algorithm maintains an actor $\mu(s|\theta^\mu)$ with a parameter θ^μ representing a deterministic policy of the agent, which deterministic mapping a state to a specific action. The critic is learned in a similar way to the Q-learning algorithm. The policy update is now changed from find maximum Q to [2]:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim p^\beta}[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}] \\ &= \mathbb{E}_{s_t\sim p^\beta}[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)}\nabla_{\theta^\mu}\mu(s|\theta_\mu)|_{s_t}]\end{aligned} \tag{6}$$

And Critic update by minimizing the loss:

$$L = \mathbb{E}[(y_t - Q(s_t,a_t|\theta_Q))^2] \tag{7}$$

where

$$y_t = r(s_t,a_t) + (s_{t+1},\mu(s_{t+1}|\theta^Q) \tag{8}$$

DDPG is a combination of algorithms with the idea of DPG and DQN algorithms to solve tasks with large state space and continuous operation space. This algorithm borrows the idea from DQN about using a replay buffer to store experience when the agent interacts with the environment, this makes for better learning because when randomly taking experience from the replay buffer to break the correlation between consecutive samples.

The next idea that DDPG borrows from DQN is to use the target network to keep learning stable. But for updating the weights for the target network, DDPG uses soft-update $\theta' = \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$ instead of directly copying the weights as in the DQN algorithm. Changing to soft-update causes the target network to change slowly, greatly improving the stability of learning.

One problem with deterministic policies is that it often quickly converges to produce the same actions while not exploring enough, thereby ignoring states that might lead to a higher total expected reward. To address this problem, the DDPG algorithm constructs an exploration policy μ' by adding noise to the actor policy

$$\mu'(s_t) = \mu(s_t) + \mathcal{N} \tag{9}$$

$\mathcal{N}$ can be anything, DDPG's author choice is to use Ornstein-Uhlenbeck process.

Algorithm 1 DDPG

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

for episode=1 to M **do**
  Get first state s_1
  Initialize a random process N for action exploration
  for t=1 to T **do**
    Select action $a = \mu(s_t|\theta^\mu) + N_t$
    Execute action a_t and receive r_t and s_{t+1}
    Store transition(s_t, a_t, r_t, s_{t+1}) in R
    Sample minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update Critic by minimizing :
    $L = \frac{1}{N}\sum_i (y_i - Q(s_i,a_i|\theta_Q))^2$
    Update Actor using policy gradient:
    $\nabla_{\theta^\mu} \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)}\nabla_{\theta^\mu}\mu(s|\theta_\mu)|_{s_i}$

    Soft update target network
    $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
    $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

B. Soft Actor-Critic (SAC)

1) Entropy: In RL, entropy refers to the predictability of the actions of an agent.This is closely related to the certainty of its policy about what action will yield the highest cumulative reward in the long run: if certainty is high, entropy is low and vice versa. With x is a random variable with probability mass function P(X) Entropy can be calculated by

$$H(X) = -\sum_{x\in X} P(x) log P(x) \tag{10}$$

In RL we want to calcutate the entropy of policy π The equation now become

$$H(\pi(.|s_t)) = -\sum_{a\in A} \pi(a|s_t) log\pi(a|s_t) \tag{11}$$

To do that we need the policy π is a Probability distribution So that the SAC actor need to be a Stochastic Actor

2) Soft actor$_c$ritic : SAC is an off-policy algorithm that is built on the actor-critic framework. The algorithm push a entropy of policy in RL objective function. The RL objective function now become

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\pi}[r(s_t,a_t) + \alpha H(\pi(.|s_t))] \tag{12}$$

The policy now need to learn how to maximize the reward and the entropy at the same time
soft actor-critic Pseudocode is

Algorithm 2 SAC

Data: $\lambda_\psi, \lambda_{\theta_1}, \lambda_{\theta_2}, \lambda_\phi, \tau$
Initialize parameter vectors $\psi, \hat{\psi}, \theta_1.\theta_2, \phi$
for each iteration **do**
  for each step in environment **do**
    $a_t \sim \pi_\phi(a_t, s_t)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $D \leftarrow D \cup (s_t, a_t, r, s_{t+1})$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_\psi \nabla_\psi J_V(\psi)$
    for i in 1,2 **do**
      $\theta_i \leftarrow \theta_i - \lambda_{\theta i} \nabla_{\theta i} J_Q(\theta_i)$
    end for
    $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi J_\pi(\phi)$
    $\psi \leftarrow \tau\psi + (1-\tau)\hat{\psi}$
  end for
end for

We will consider some neural network to describe: state value function: $V_\psi(s_t)$ soft Q function: $Q_\theta(s_t, a_t)$ policy: $\pi_\phi(a_t|s_t)$ The parameters of these network are ψ, θ, ϕ.
The soft value function is trained to minimize the squared residual error

$$J_v(\psi) =_{s_t \sim D} [1/2(V_\psi(s_t) -_{a_t \sim \pi_\phi} [Q_\theta(s_t, a_t) - log\pi_\phi(a_t|s_t)])^2] \tag{13}$$

The gradient of equation 9 can be estimated by

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t, a_t) + log\pi_\phi(a_t, s_t)) \tag{14}$$

The soft Q-function parameters can be trained to minimize the soft Bellman residual

$$J_Q(\theta) =_{(s_t,a_t)\sim D} [1/2(Q_\theta(s_t, a_t) - \hat{Q}(s_t, s_t))^2] \tag{15}$$

with

$$\hat{Q}(s_t, s_t) = r(s_t, s_t) + \gamma_{s_t+1 \sim p}[V_{\hat{\psi}(s_t,a_t)}] \tag{16}$$

equation 11 can be optimized with stochastic gradients

$$\nabla_\theta J_Q(\theta) = \nabla_\theta Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_{\hat{\psi}(s_{t+1})}) \tag{17}$$

Finally, the policy parameters can be learned by directly minimizing the objective function

$$J_\pi(\phi) =_{s_t \sim D, \epsilon \sim N} [log\pi_\phi(f_\phi(\epsilon, s_t)|s_t) - Q_\phi(s_t, f_\phi(\epsilon_t; s_t))] \tag{18}$$

We can approximate the gradient with

$$_\phi J_\pi(\phi) = \nabla_\phi log\pi_\phi(a_t, s_t) \tag{19}$$
$$+(\nabla_{a_t} log\pi_\phi(a_t, s_t) - \nabla_{a_t} Q(s_t, a_t))\nabla_\phi f_\phi(\epsilon; s_t) \tag{20}$$

C. Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 stands for Twin Delayed Deep Deterministic Policy and is an off-policy actor-critic algorithm. It addresses the overestimate bias problem that existed in DDPG and first SAC version.

Overestimation bias is when the critic highly rates suboptimal actions and the actor learns to increase probabilities to choose those actions. This is problematic, because the actor is trained towards a suboptimal policy, then picking a bad action leads to the critic learning ineffectively. That is a bad loop to have a good policy. To address this problem, TD3 developed the idea of using two Q networks of Double Q-Learning and calculating Q target update with:

$$y = r_t + \gamma \min_i Q(s_{t+1}, a_{t+1}) \tag{21}$$

In addition, TD3 also proposes a delayed policy update. The approximation function always has estimation errors, the actor-critic technique accumulates them:

$$\begin{aligned} Q_\theta(s_t, a_t) &= r_t + \gamma \mathbb{E}\left[Q_\theta(s_{t+1}, a_{t+1})\right] - \delta_t \\ &= r_t + \gamma \mathbb{E}\left[r_{t+1} + \gamma \mathbb{E}\left[Q_\theta(s_{t+2}, a_{t+2}) - \delta_{t+1}\right]\right] - \delta_t \\ &= \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}\left[\sum_{i=t}^{T} \gamma^{i-t}(r_i - \delta_i)\right] \end{aligned} \tag{22}$$

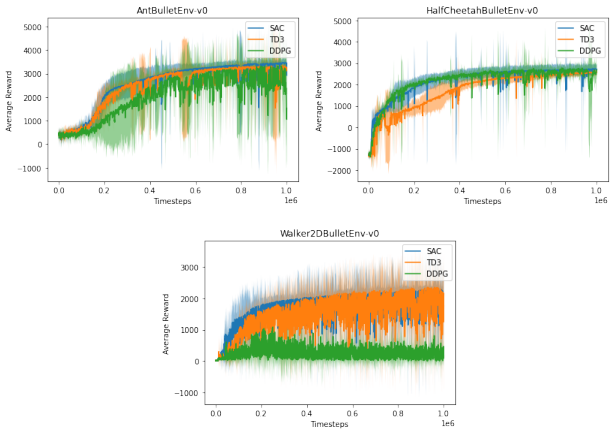
To reduce approximation errors, TD3 updates the actor only with each d step update critics, then softly updates to the actor and critics target networks with the parameter τ. That helps critics have more time to converge and reduce the estimation error before using it to update the actor.

Algorithm 3 TD3

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

V. Experiments



These pictures show the average reward of three algorithm with three environment HalfCheetahBulletEnv-v0, AntBulletEnv-v0 and Walker2DBulletEnv-v0. With 4 random seed the Experiments show that SAC work very good for all of these environment. SAC and TD3 success to convert for all environment and DDPG success to convert only two of them. In the AntBulletEnv-v0 environment, you can see the problem of DDPG - overestimation bias. from 200000 timesteps to 400000 timesteps the reward of DDPG is very unstable because of the problem.
In the HalfCheetahBulletEnv-v0 environment, DDPG is likely bester than TD3 which is realy unusual. It is because the problem of TD3 underestimation bias happen and DDPG overestimation bias problem didn't. It make the performance of DDPG bester than TD3
In the Walker2DBulletEnv-v0 environment, reward is very unstable. But it show that SAC work much more efficiently than the other.

VI. Future Work

All three off-policy algorithms for continuous control we describe can be used to solve continuous control problems, but they have slow convergence and are greatly influenced by hyperparameters. We found that reinforcement learning algorithms can combine with other methods like Genetic Algorithm or Evolutionary Strategy (ES) can learn better and outperform the original deep reinforcement learning algorithms (DRL). So in the future, we want to learn about some methods that combining by ES and DRL.

VII. Conclusion

In this paper, we described three off-policy algorithms for continuous control tasks. The algorithm DDPG combines the idea of DPG algorithm and the success of the DQN algorithm to expand to enable to solve tasks have high-dimensional in both action space and state space. TD3 improves the DDPG by addressing the overestimate bias problem that existed in DDPG and the first SAC version. SAC adding the entropy term in reward function and using a stochastic policy rather than a deterministic policy like DDPG and TD3 using. The Agent now need to learn how to maximize the reward and the entropy at the same time. The stochastic policy and Entropy also make the agent learn many optimal solutions for the same problem.

Acknowledgment

The authors would like to thank Dr. Luong Ngoc Hoang for dedicated guidance.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning*.
[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

Huynh Hoang Vu graduated from Mac Dinh Chi High School in Ho Chi Minh City. He is studying 2rd year at University of Information Technology - VNUHCM. He belongs to the Department of Computer Science, class KHTN2020.

Truong Mai Tan Luc started study a bachelor's degree at the University of Information Technology - VNUHCM, currently a second-year student in class KHTN2020 with a major in Computer Science.

Tran Huu Khoa started study a bachelor's degree at the University of Information Technology - VNUHCM, currently a second-year student in class KHTN2020 with a major in Computer Science. Graduated from high school at THPT Chuyen Phan Ngoc Hien in Ca Mau