

UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE

Deep Deterministic Policy Gradient (DDPG) & Twin-Delayed DDPG (TD3)



Instructor: Luong Ngoc Hoang

Student: Ha Huy Hoang - 22520460

Class: CS211.P11

Table of Contents

1. Introduction	1
2. Statistics	2
2.1. HalfCheetah-v4	3
2.2. Ant-v4	4
3. Conclusion	5

1. Introduction

DDPG^[1] (Deep Deterministic Policy Gradient) is an actor-critic algorithm that extends DQN to handle continuous action spaces. It uses deterministic policy gradient theorem to train a deterministic policy. DDPG employs two networks: an actor network that outputs actions and a critic network that estimates Q-values. Like DQN, it uses experience replay and target networks to stabilize training, but adds noise to actions during exploration.

The actor and critic updates in DDPG pseudocode follow:

Algorithm 1 Deep Deterministic Policy Gradient

```

1: Input: initial policy parameters  $\theta$ , Q-function parameters  $\phi$ , empty replay buffer  $\mathcal{D}$ 
2: Set target parameters equal to main parameters  $\theta_{\text{targ}} \leftarrow \theta$ ,  $\phi_{\text{targ}} \leftarrow \phi$ 
3: repeat
4:   Observe state  $s$  and select action  $a = \text{clip}(\mu_{\theta}(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$ , where  $\epsilon \sim \mathcal{N}$ 
5:   Execute  $a$  in the environment
6:   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
7:   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$ 
8:   If  $s'$  is terminal, reset environment state.
9:   if it's time to update then
10:    for however many updates do
11:      Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:      Compute targets


$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$


13:      Update Q-function by one step of gradient descent using


$$\nabla_{\phi} \frac{1}{|B|} \sum_{(s, a, r, s', d) \in B} (Q_{\phi}(s, a) - y(r, s', d))^2$$


14:      Update policy by one step of gradient ascent using


$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi}(s, \mu_{\theta}(s))$$


15:      Update target networks with


$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1 - \rho) \phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned}$$


16:    end for
17:  end if
18: until convergence

```

TD3^[2] (Twin Delayed Deep Deterministic Policy Gradient) is an enhancement to DDPG that addresses overestimation bias and improves training stability. It introduces three major modifications: clipped Double-Q Learning, delayed policy updates and target policy smoothing. The clipped Double-Q Learning using the smaller Q-value for the target, and regressing towards that, helps fend off overestimation in the Q-function; while delayed policy updates and noise addition to target actions further stabilize training.

TD3 concurrently learns two Q-functions, Q_{ϕ_1} and Q_{ϕ_2} , by mean square Bellman error minimization, in almost the same way that DDPG learns its single Q-function.

- **Target Policy Smoothing:**

- Adds clipped noise to target actions: $\mu'(s') + \epsilon$, where $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$.
- Acts as a regularizer to prevent the policy from exploiting incorrect peaks in the Q-function.
- Smooths out the Q-function over similar actions for more stable learning.

- **Clipped Double-Q Learning:**

- Uses two Q-functions instead of one.
- Takes the minimum Q-value between the two critics for target calculation:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_i, \text{targ}}(s', a'(s')),$$

[1]: Deep Deterministic Policy Gradient

[2]: Twin Delayed DDPG

- Both Q-functions are trained using this same target.
- Helps prevent overestimation bias common in DDPG.

- **Delayed Policy Updates:**

- Policy network is updated less frequently than Q-functions.
- Typically updated once for every 2-3 Q-function updates.
- Reduces volatility in training by allowing Q-functions to converge more before policy updates.

Algorithm 1 Twin Delayed DDPG

```

1: Input: initial policy parameters  $\theta$ , Q-function parameters  $\phi_1, \phi_2$ , empty replay buffer  $\mathcal{D}$ 
2: Set target parameters equal to main parameters  $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$ 
3: repeat
4:   Observe state  $s$  and select action  $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$ , where  $\epsilon \sim \mathcal{N}$ 
5:   Execute  $a$  in the environment
6:   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
7:   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$ 
8:   If  $s'$  is terminal, reset environment state.
9:   if it's time to update then
10:    for  $j$  in range(however many updates) do
11:      Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:      Compute target actions
          
$$a'(s') = \text{clip}(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

13:      Compute targets
          
$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

14:      Update Q-functions by one step of gradient descent using
          
$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2 \quad \text{for } i = 1, 2$$

15:      if  $j \bmod \text{policy\_delay} = 0$  then
16:        Update policy by one step of gradient ascent using
          
$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

17:        Update target networks with
          
$$\begin{aligned} \phi_{\text{targ},i} &\leftarrow \rho \phi_{\text{targ},i} + (1 - \rho) \phi_i \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned} \quad \text{for } i = 1, 2$$











18:      end if
19:    end for
20:  end if
21: until convergence

```

2. Statistics

As the teacher has guided, I will use TensorBoard to compare different runs. However, there is an issue with large fluctuations between steps, making the graphs difficult to read and analyze. Therefore, I have adjusted the smoothing to 0.99 to make the analysis easier. Since the second neural network of TD3 is only used for cross-checking with the first neural network, helping to evaluate the value of actions more accurately, it has no value in analysis as it is not compared with DDPG (using the first neural network for comparison, therefore in the report will not comment on these graphs).

Each color lines presents a algorithm with seed that have the structer likes algorithm_seed, the seeds and algorithm i used for experiment is listed below:

	DDPG_22520460		TD3_22520460
	DDPG_22520461		TD3_22520461
	DDPG_22520462		TD3_22520462
	DDPG_22520463		TD3_22520463
	DDPG_22520464		TD3_22520464

2.1. HalfCheetah-v4

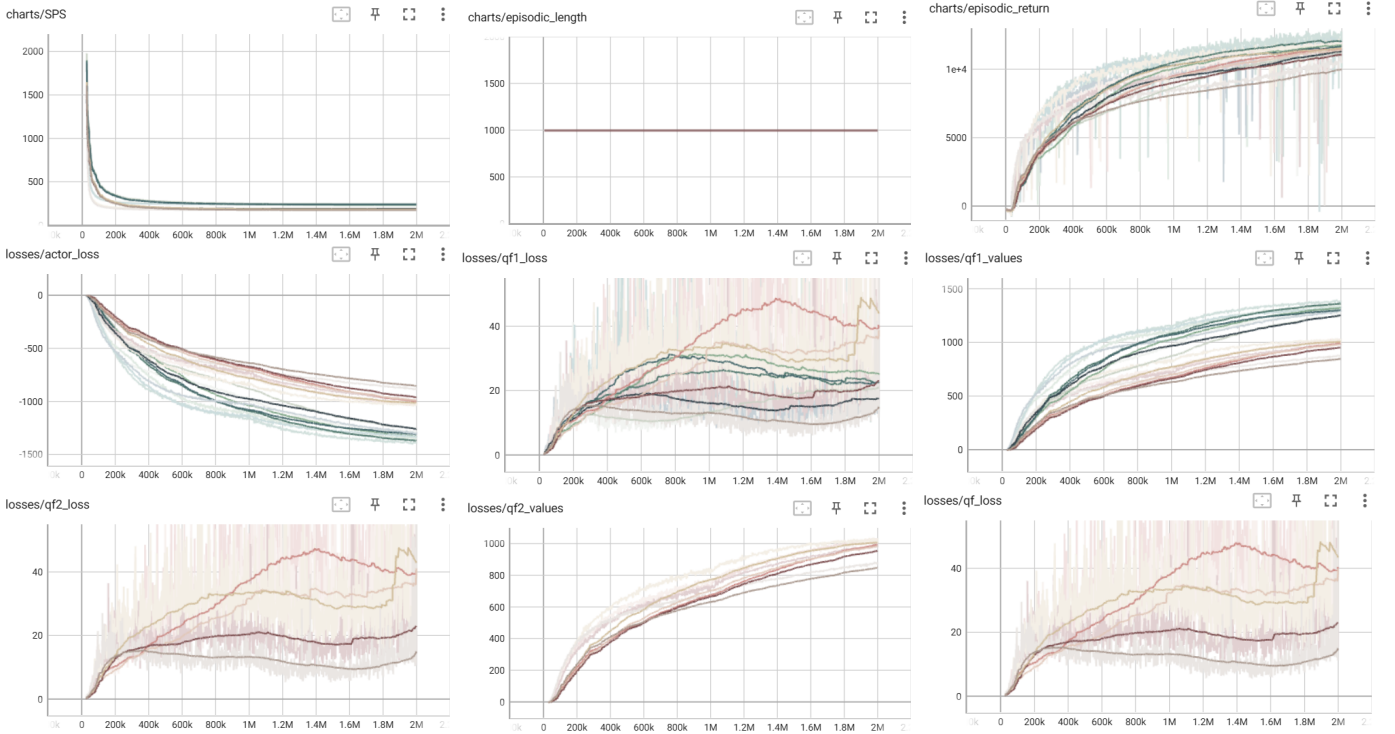


Figure 1. HalfCheetah-v4

Looking at the first chart, we can see that the SPS of TD3 is consistently lower than that of DDPG. The SPS of TD3 ranges from approximately 180 to 200, while DDPG's ranges from 240 to 250 (both trained on the same NVIDIA TESLA P100 GPUs configuration on Kaggle). TD3 has a lower SPS compared to DDPG because it uses a more complex architecture with two Critics networks (instead of one like DDPG), along with mechanisms such as delayed updates and target policy smoothing, resulting in more computations required for each training step.

The episodic length just witnessed one value 1000 because the episode truncates when the length of the episode is greater than 1000^[3]. In terms of episodic return, both show an upward trend in scores (return) over the course of training.

TD3 has a higher actor loss because it updates the actor less frequently (delayed policy update) and uses $\min(Q_1, Q_2)$ for policy updates, resulting in more cautious learning. In this case, based on the rewards graph (qf1_values), it shows that DDPG (blue) achieves higher values compared to TD3 (red). The mechanisms in TD3 to prevent overestimation (using min Q-values and target policy smoothing) may be overly cautious and unnecessary in this specific training environment, leading to lower performance than DDPG (in some seed TD3 shows the much higher qf1_loss than DDPG). This indicates that although TD3 is designed to address the shortcomings of DDPG, these improvements do not always result in better performance in every environment.

[3]: [Half Cheetah - Gym Documentation](#)

2.2. Ant-v4

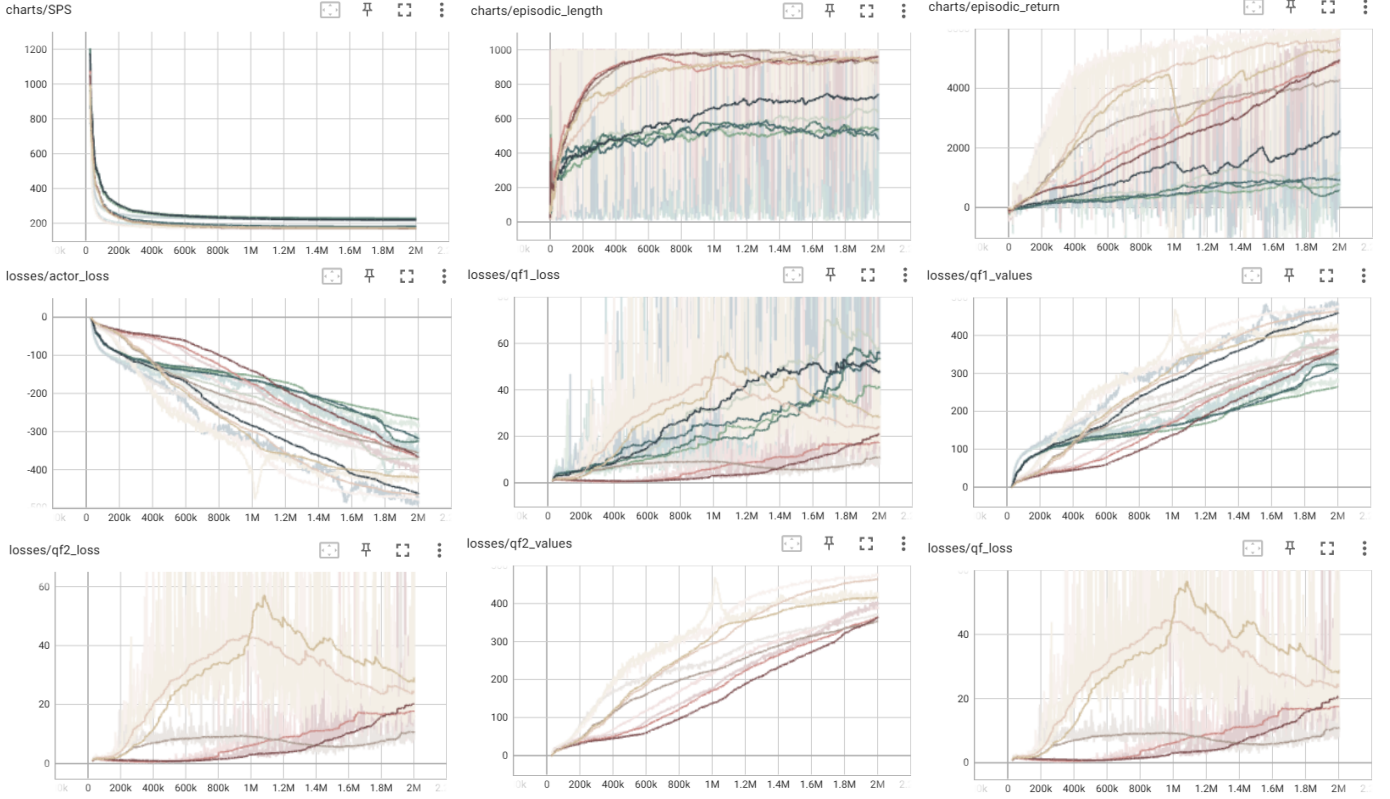


Figure 2. Ant-v4

As mentioned in the earlier context, TD3 consistently has a lower SPS compared to DDPG. This is due to its more intricate architecture featuring two Critics networks (unlike DDPG, which uses only one), along with techniques such as delayed updates and target policy smoothing, leading to increased computational demands for each training step.

However, in the Ant-v4 environment, we observe a difference in the episodic length and episodic return produced by the two algorithms. In both charts representing these values, TD3 consistently shows higher values, indicating superior performance in this environment. Specifically, the episodic length of TD3 converges much faster than DDPG, reaching 1000^[4] (the termination condition of the environment if the Ant is considered "healthy") without smoothing, while DDPG fluctuates between 400 and 750. Additionally, the rewards returned by TD3 are significantly higher than those of DDPG (ranging from 4000 to 5800 compared to 600 to 2500). From this, we can conclude that TD3 outperforms DDPG in the Ant-v4 environment.

In terms of actor_loss, both show a downward trend in loss over the course of training. In contrast, the qf1_loss of TD3 becomes lower than that of DDPG toward the end of training. In the qf1_values chart, we observe that TD3 has higher values compared to DDPG in most seeds. This may be because, in a complex environment like Ant-v4, DDPG is more prone to getting stuck in local optima due to overestimation, whereas TD3's more cautious mechanisms (min Q-values, delayed policy update) help it discover better policies.

[4]: [Ant - Gym Documentation](#)

3. Conclusion

All the algorithms and seed i runned with this tempalte command:

```
python cleanrl/algo.py --seed seed --env-id game --total-timesteps 2000000 --capture_video
```

There is an issue with capturing videos where only the first video is recorded properly, and the remaining videos appear black. I searched for solutions online but couldn't find any, so I added an episodic trigger to capture a video at episode 1999. Why episode 1999? Because the total-timesteps I used is 2,000,000, and the maximum timesteps for each episode in both environments is 100,000. Thus, I chose to record the video at the final episode in the hope that it would capture the best performance. The best videos of each algorithm in environments are stored in this [link](#).

In less complex environments like HalfCheetah-v4, the performance charts of DDPG and TD3 are almost equivalent. However, based on my subjective evaluation, the result videos of DDPG across all seeds are not as good. This is because agents in DDPG tend to run too fast and sometimes stumble, whereas TD3 does not exhibit this issue.

In more complex environments like Ant-v4, DDPG's performance charts and result videos are significantly worse compared to TD3. In some videos, the agents using DDPG run normally for a short distance and then flip over, remaining that way for the rest of the timesteps (as the center height z does not reach the threshold, so it isn't truncated)(I will choose the best video based solely on the agent's running time without considering the timesteps). In other cases, the agents become "unhealthy" during the run and are stopped (and from a personal perspective, the agent moves quite slowly and a relatively short distance compared to TD3). On the other hand, in TD3's videos, the agents consistently run normally without such problems.

In summary, we can trade off the relatively small differences in the performance of the two algorithms in less complex environments like HalfCheetah-v4 for faster training by using DDPG. However, in more complex environments like Ant-v4, choosing TD3 is more reasonable because the training time is not significantly different, but the performance is substantially better compared to DDPG. A comprehensive list of results performed by the authors of the TD3 paper is available on page 7 of the [paper](#)