Application of the twin-delayed deep deterministic policy gradient algorithm in the BipedalWalker environment

Anonymous author

Abstract

This paper aims to apply the twin-delayed deep deterministic policy gradient algorithm for learning to walk in both the BipedalWalker v3 and BipedalWalker v3 hardcore environments. The paper introduces the enhancements of the TD3 algorithm compared with the DDPG algorithm. Additionally, the network's update process is explained using flowcharts and equations, and various modifications were made to the basic model to improve its performance in the BipedalWalker environment. The results from both environments demonstrate great convergence after training. After that, the limitation and future work of the current project are discussed.

1 Methodology

The twin-delayed deep deterministic policy gradient[2] reinforcement learning method is applied to train the bipedal robot to walk in the virtual environment. BipedalWalker v3 and BipedalWalker v3 hardcore[1] are the training environment, including a 24-dimensional observation space and a 4-dimensional action space representing the state data and action data of the bipedal walker, respectively. TD3 algorithm is an improved deep deterministic policy gradient algorithm[3] which proposed double Q-learning to alleviate the overestimated Bias estimates problem. There are three critical improvements in the TD3, Clipped Double-Q Learning, 'Delayed' Policy Updates and Target Policy Smoothing. DDPG algorithm introduces deep learning into the reinforcement learning to use a deep neural network building the policy, which makes it display outstanding performance on the high-dimensional state and action spaces compared to the traditional reinforcement learning method. However, the overestimation bias tends to appear in the Actor-Critic structure of the DDPG algorithm. It results in the model being stuck in a suboptimal state, being unable to learn a better action.

Follow this motivation, TD3 algorithm proposes two Q-learning are used to calculate the Q value based on the same parameter. The lower Q values are selected to calculate the error to reduce the impact of the overestimate.

Secondly, 'Delayed' Policy Updates allow the policy and target network update frequency to be lower than the Q-functions. This way, the volatility that typically occurs in DDPG due to the policy update can be filtered efficiently.

Thirdly, to alleviate the over-fitting of the valuation function, the clipped noise with a slight variance is added to the action. It encourages the target policy to update smoothly to avoid the appearance of the incorrect peak and stabilise the entire convergence process.

1.1 Algorithm process

The update process of TD3 is similar to the DDPG; There are six networks in the TD3 and two more than in DDPG. In the training process, the first step is initialising the Actor and Critic. Then substitute the parameters into the actor and critic in the target network to complete the initialisation of the six networks. Next, make the actor interact with the environment and get the action a according to the state s. Based on the action a, the

next state $s^{'}$ and reward r will be returned by the environment. In this way, different $s, a, r, s^{'}$ are stored in the buffer as the dataset used in the critic training. Two Q values are generated by two critics in the condition of s and a. $s^{'}$ will be the state of the target actor and output the add-noise action $a^{'}$ (noise is added to all dimensions of the action) as equation 1 shown.

$$a^{'}(s^{'}) = clip(\mu_{\theta_{targ}}(s^{'}) + clip(\epsilon, -c, c), a_{Low}, a_{High}), \epsilon \sim \mathcal{N}(0, \sigma) \tag{1}$$

Enter $s^{'}$ and a' into two target critics and two $Q^{'}$ of the target network are calculate according to the equation 2. The minimal of them are selected as the label of the critic network.

$$Q^{'} = r + \gamma min_{i=1,2}(Q_{\theta_i}(s^{'}, \pi_{\phi_1}(s^{'}))) \tag{2}$$

Then, update the critics to minimise the difference L between evaluation value Q and target value $Q^{'}$ using the gradient descent algorithm.(equation 3)

$$argmin_{\theta_i} N^{-1} \sum (Q^{'} - Q_{\theta}(s, a))^2 \tag{3}$$

Due to the 'Delayed' Policy update, the target actor and target critics networks are updated after the actor and critics networks d step updating following the equation 4. In the update process of the target network, apply the soft update in the target network. There is a τ representing the learning rate in equation 5 and equation 6, which is used to implement the weighted average of the previous network parameters and the corresponding network's new parameters. At last, the weighted parameters are assigned to the target network performing the update.

$$\nabla_{\phi} J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_{\phi}(s)} \nabla_{\phi} \pi_{\phi}(s) \tag{4}$$

$$\theta^{Q_i^{'}} = \tau \theta^{Q_i} + (1 - \tau) \theta^{Q_i^{'}} (i = 1, 2) \tag{5}$$

$$\theta^{\mu^{'}} = \tau \theta^{\mu} + (1 - \tau) \theta^{\mu^{'}} \tag{6}$$

Then iterate the model by the loop of equation 1-6, and the entire update process of the networks can be described as the figure 1.

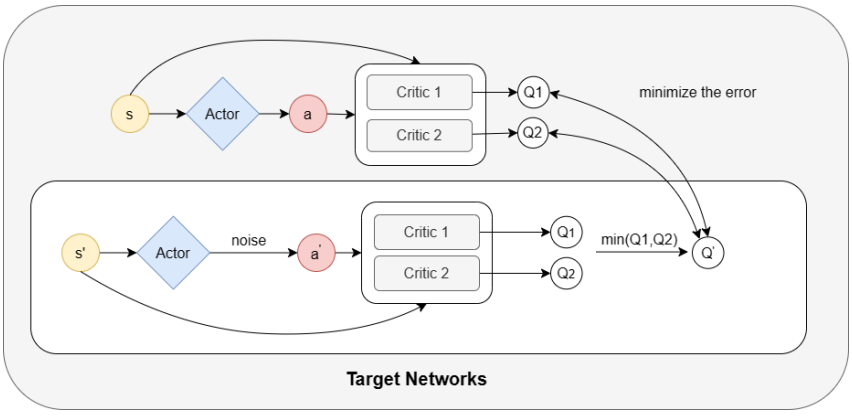

Figure 1: Flowchart of TD3

1.2 Improvement

To obtain better convergence results, the different parameter sets are applied in the default model[4]. The suitable combination of parameters leads to an improvement in learning ability so that reward of the model reaches 300 after hours. However, the tuned model is still in negative reward for the first few hours and shows high volatility even in the late stages of convergence. To further improve the learning efficiency, there are three improvement methods applied in the model according to the environment features.

The first method is the adjustment of the reward of the fall. In the default environment, the robot gets a -100 reward if it falls. The punishment for falls is much more severer than the usual action, which leads to the model's tendency to learn how to avoid falls instead of walking. Hence, the fall action is assigned to the normal reward interval $[-1, 1]$. In the improved model, if the robot falls, `reward = -1`.

Secondly, classify the state of *done*. There are three conditions that make *done* to be $True$, "time out", "Pass," and "fall". The Q value will be ignored when *done* is $True$ in a state. However, "time out" and "pass" not means that the state is as worse as the "fall". Therefore, to store more valuable data in buffer, *done* is only set to $True$ when the robot falls so that more Q values can be used in training.

The third method is dynamic noise. The noise is significant in the early stages of training to smooth the learning curve. However, the constant noise causes a negative effect on the convergence of the algorithm in the later stages of the training, which even causes a "reward cliff" in the training process. The Noise decay rate is introduced so that noise decreases as the training progresses, which ensures stability and convergence in the later stages. In addition, the different sizes of the network are tried in the v3 and Hardcore-v3. It founds that the network size should increase as the environmental difficulty increases. However, too large a network size can have a negative impact on the convergence of the model. Hence, the network width of v3 and Hardcore-v3 is set to 120 and 190, respectively.

2 Convergence Results

The result of the default TD3 model in the BipedalWalker v3 environment is shown in figure 2. Although the learning curve shows a tendency to converge, the scoring was consistently stuck below 300 and unstable. Figure 3 is the result of the improved TD3 algorithm showing great convergence and scores over 300 consistently after 1100 epochs. It indicates that improved TD3 algorithms have passed the v3 environment. The episode reward has been over 0 before 100 episodes and stabilised at over 310 after 1500 episodes reaching the highest reward, 318.6 at epoch 1573.

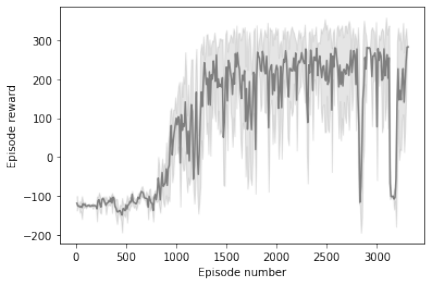

Figure 2: Default TD3

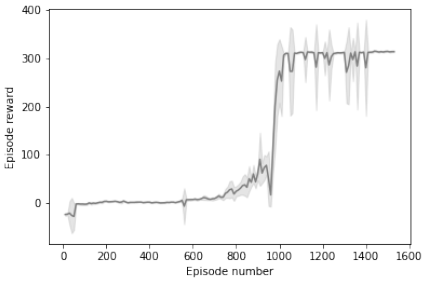

Figure 3: Improved TD3

Switch the environment to the BipedalWalker v3 hardcore and increase the network size to 190 to match the more complex environment. The learning curve is shown in figure 4. It takes 4000 more episodes to converge compared with the BipedalWalker v3 environment. After about 5000 episodes, the model floats around 280 and achieves convergence. The

average reward did not further increase to 300 in the following training, but the video represents the robot can pass the hardcore environment stably.

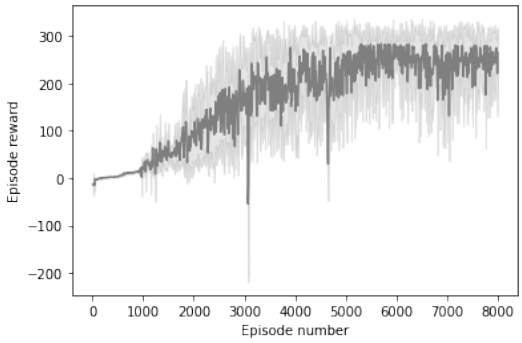

Figure 4: Improved TD3 (BipedalWalker v3 hardcore)

3 Limitations

The TD3 algorithm displays outstanding performance implementing the convergence on BipedalWalker v3 and hardcore environment. However, for the BipedalWalker v3 environment, the model still spent nearly 1000 episodes stuck nearby 0 rewards. In the environment of BipedalWalker v3 hardcore, the learning curve is more smooth. However, it shows more apparent fluctuations and does not reach 300 rewards during the training. By analysing the generated video, there are many redundant actions when the Bipedal robot tackles the different task in environment, which causes the deduction of the reward. Secondly, the training process of v3 hardcore costs over 10 hours but remains showing a unstable convergence.

Future Work

In the future, more parameters will be tested to obtain better performance and try to develop more tricks based on the algorithm until it can generate an excellent convergence curve in the hardcore environment. In addition, the TD3 algorithm would be applied in more environments that required continuous control, such as Car Racing and Lunar Lander environments from Open AI. Compare the result and time spent in different environments to determine robust of the TD3 algorithm.

References

[1] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[2] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[3] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[4] XinJingHao. *Xinjinghao/TD3-Pytorch: A clean and robust PYTORCH implementation of TD3 on continuous action space*. 2022. URL: `https://github.com/XinJingHao/TD3-Pytorch`.