Learning to Walk Randomly

Anonymous author

Abstract

In this task, I use the Twin-Delayed DDPG (TD3) algorithm as a basis to train a small bipedal robot to learn to walk efficiently. First, we train in a normal environment and optimize the training strategy, which performs well; then we train in a difficult scenario to verify the reliability of our improved algorithm. reliability of our improved algorithm.

1 Methodology

First, using the objective network to reduce the error accumulation to decrease the variance; second, updating by delaying the strategy until the estimates converge; and finally, using a double-critic update strategy to use the higher valuation of the two as an approximate upper bound for the true value estimate to achieve the effect of underestimation.

Strategy network, Q network and its target network In the TD3 algorithm, 1 strategy neural network $\mu(\phi)$ and 2 evaluation neural networks Q are used to represent deterministic strategy gradient and state action value functions, respectively. In addition, to increase the stability of the network, the concept of target network is introduced with reference to the DQN algorithm, corresponding to 1 target strategy network and 2 target evaluation networks. The policy network inputs state value s and outputs a determined action a : The

$$a = \pi_{\varphi}(s) + \varepsilon \tag{1}$$

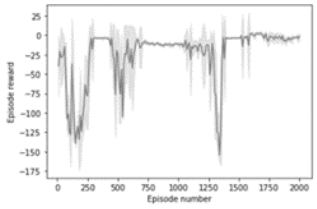
where: ε is the random noise, and $\varepsilon \sim N(0, \sigma)$, is used for random exploration during learning. Perform the action a and obtain the reward value r and the new observation S' , store the meta-array (s, a, r, s') stored in the experience pool; a mini-batch is randomly selected from the experience pool of (s, a, r, s') , 2 Q-network output functions $Q_{\theta_i'}(i = 1, 2)$, taking the smaller value of the 2 Q-networks and calculating $Q_{\theta}(s, a)$ The target values of are shown below:

$$y^{\text{target}} = r + \gamma \min_{i=1,2} Q_{Q_i'}(s', \tilde{a}) \tag{2}$$

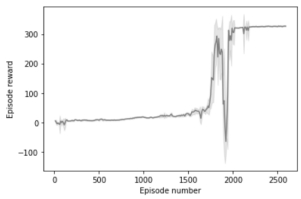
where: the target action (the action in the input target Q network) is $\tilde{a} = \pi_{\varphi'}(s') + \tilde{\varepsilon}$, adding the cropped noise to the target action $\tilde{a}$ that will make the target action $\tilde{a}$ near the original action, and $\tilde{\varepsilon} \sim \text{clip}(N(0, \tilde{\sigma}))$. After calculating the target valuey $y\,(r, s')$ after minimizing the $Q_{\theta_1}, Q_{\theta_2} y^{\text{target}}$ of the mean square error to learn the parameters of the evaluation network.

2 Convergence Results

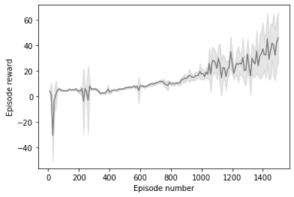
BipedalWalk-v3:The results for the bipedal robot trained in the BipedalWalk-v3 environment are generally feasible, especially after implementing the two tricks. We first tried to train using the basic algorithm, but as can be seen from the figure, there was no convergence and the reward was very low. Therefore, we considered a reward mechanism that would enable the robot to learn to walk continuously.



In the second experiment, we improved the algorithm by manually setting done to True when the robot falls, so that the Q value of the subsequent states is initially assumed to be zero and the reward is subtracted by 1 when the robot is found to have fallen. the graph shows the dramatic impact, as the robot is able to achieve a score of around 300 and converge.



BipedalWalkerHardcore-v3:In addition, testing our results in difficult mode revealed that the robot could not handle large obstacles well, so to solve this problem, we used a smaller learning rate based on the previous one. The results are shown in the figure, and the robot was able to go through the whole process within 3000 training cycles.



The effect is shown in the figure.

3 Limitations

The two major limitations of this experiment lie in the training time and the limitation of the algorithm itself. The training time of a single conventional experiment may take up to 4 hours, and the training of a slightly more complicated experimental code takes longer, maybe 10 hours. Already, this leads to not much time to try other hyperparameters or better parameters; followed by the limitation of the TD3 model itself, although the TD3 algorithm is already an improved version of the DDPG algorithm, but due to our limited training time, there is no Try other algorithms that are more suitable for the model, so the experimental results may also be affected by the algorithm itself.

Future Work

If time is limited in the future, we will spend more time trying different hyperparameters and may try other algorithm models,such as SAC[1].

References

[1] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.