Learning to Walk with TD3-FORK

mccb22

Abstract

This paper proposes training a bipedal robot to walk using an extension of the Twin Delayed DDPG (TD3) Reinforcement learning algorithm, known as TD3-FORK. It is an Actor-Critic method that uses a different kind of actor, a FORK that forecasts future states and rewards to improve its policy. Using this algorithm, a bipedal robot is successfully trained to walk in the OpenAI gym environment considerably faster and in less episodes than the standard TD3 algorithm

1 Methodology

TD3-FORK is used in the BipedalWalker-v3 environment to successfully induce a bipedal robot to walk

1.1 Bipedal Walker

The main environment used in this paper is OpenAI gym's BipedalWalker-v3 [2]. The goal of this environment is to get a bipedal robot to walk, hence the robot gets a -100 negative reward for falling and a positive reward for moving forward, totalling 300+ points if it reaches the far end. To achieve this the bipedal walker can apply a continuous amount of torque in the range of (-1, 1) to the 2 joints on each of its legs. The state space in which it can base these actions on includes hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar readings.

1.2 DDPG

Deep deterministic policy gradient is an actor-critic reinforcement learning algorithm. Actor-critic algorithms involve an actor that takes states and predicts the action that maximises long-term reward, and a critic that takes state-action (s,a) pairs and returns the expected long term reward or Q value [4]. These Q values are learnt using the right hand side of the Bellman Equation for the optimal value function Q* or Q_{targ}:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}[r(s,a) + \gamma \max_{a'} Q^*(s',a')] \tag{1}$$

Where s' and a' are the next state and action at time t+1, r(s,a) is reward and gamma is the discount factor applied to future rewards. Meaning critics output Q values that approximate the target Q values given by the Bellman Equation. Therefore we want to minimise the mean-squared Bellman error, a measure of how close our Q or critic function Q(s,a) is to satisfying the Bellman equation.

$$L(D) = \mathbb{E}_{(s,a,r,s',d) \sim D}[Q(s,a) - (r + \gamma(1-d)Q_{targ}(s', \mu_{targ}(s')))^2] \tag{2}$$

(s,a,r,s',d) are sampled from the set of possible transitions d, and μ_{targ} is the target policy. Importantly, having the optimal Q function Q* allows us to obtain the optimal action given a state:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{3}$$

and therefore the optimal actor and policy.

In the continuous action space the assumption can be made that Q* is differentiable with respect to the action, and therefore we can approximate optimal action with a gradient based learning rule for the policy. The actor parameters are updated in the following way with β as the learning rate:

$$\phi \leftarrow \phi + \beta \nabla_{\phi} Q(s, A_{\phi}(t)) \tag{4}$$

The actor, critic and their targets are parameterized by deep neural networks, making DDPG a deep reinforcement learning algorithm.

1.3 TD3

Twin Delayed DDPG is an extension of deep deterministic policy gradient that counteracts DDPG's tendancy to overestimate Q values and break policy [3]. In TD3 there are:

1) 2 critics that produce 2 Q values, of which the smaller is used to form the targets in the mean squared Bellman error loss function.

2) Delayed policy updates, the actor is updated less frequently than the critics

3) Target policy smoothing adds noise to the target action, acting as a regularizer.

My implementation of TD3 is based off code at [1]. I merge this code with logging code from the template. Then I also add initial random exploration for the first 12 episodes of training, and a procedure that decays exploration noise once the average of the most recent 10 episodes reaches a reward of over 300. This encourages initial random exploration to increase the probability of finding successful strategies, then prioritises exploitation once these successful strategies are found.

1.4 TD3-FORK

TD3-FORK is an extension to TD3 that uses a FORK- a forward looking actor for model free reinforcement learning. This allows forecasting of future states and their values to improve the policy [6].

2 additional neural networks are added in TD3-FORK, a system network that predicts the next state given a state-action pair and a reward network that predicts the reward given a state-action pair and the next state. Both are trained using mini-batches from the replay buffer, with smooth-L1 loss and mean-squared error loss respectively.

These additional networks aid in updating the actors parameters, giving a different update procedure in TD3-FORK compared to TD3:

$$\phi \rightarrow \phi + \beta(\nabla_{\phi} Q(s, A_{\phi}(s)) + \nabla_{\phi} R_{\eta}(s, A_{\phi}(s)) + \gamma \nabla_{\phi} R_{\eta}(s', A_{\phi}(s')) + \gamma^2 \nabla_{\phi} Q(s'', A_{\phi}(s''))) \tag{5}$$

Where R_{η} is the reward network and s' and s" are future states forecast by the system network.

The following extra terms are also added to the loss function when training the actor:

$$wR_{\eta}(s, A_{\phi}(s)) + w\gamma R_{\eta}(s', A_{\phi}(s')) + w\gamma^2 Q(s'', A_{\phi}(s''))) \tag{6}$$

This results in an actor that can "plan" and perform better actions, by using predicted future states and values as well as its current state and reward.

FORK also uses adaptive weight to increase exploration in early episodes and increase exploitation in later episodes. This is the w term in the above loss function that scales all the added terms from the reward and system network. $w = (\frac{r'}{r_0})_0^1 w_0$ where r' is the moving

average of cumulative reward, r_0 is the reward when the goal is reached and w_0 is the initial weight.

I extend my TD3 code to TD3-FORK based on github repository [5], that was linked in the paper "FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning" [6].

2 Convergence Results

The model consistently converges on scores above 300 in the bipedal walker environment.

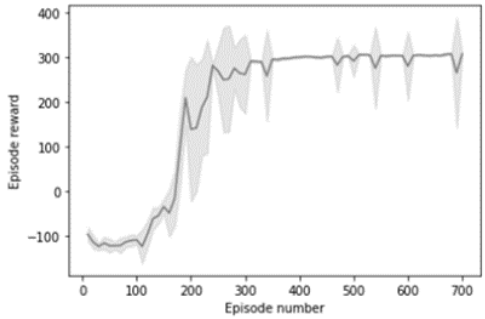

Figure 1: Convergence results for TD3-FORK. Episode reward is average of the last 10 episodes reward, it is recorded every 10 episodes

As shown in figure 1 episode reward starts to rapidly increase around episode 160, reaching average rewards of almost 300 around episode 240. In episodes beyond this the average reward starts to stabalise and slightly increases to consistently over 300 by the 400th episode.

Although the exact number of episodes it takes the reward to converge towards 300 varies between random seeds, the model is consistently able to reach scores of 300+ by episode 500, usually sooner.

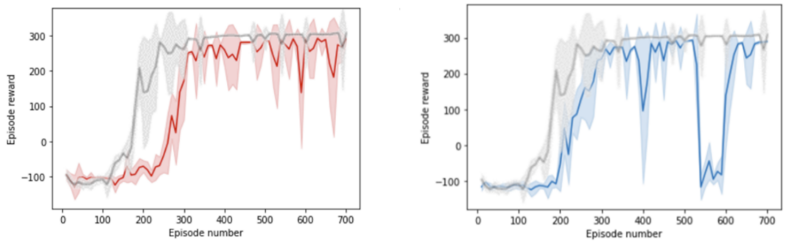

Figure 2: (a) Convergence results for TD3 (red) compared to TD3-FORK (grey). (b) Convergence results for TD3-FORK with (grey) and without (blue) initial random exploration and noise decay

Figure 2a shows a comparison of the convergence of TD3 and TD3-FORK. TD3-FORK reaches higher scores towards 300 about 75 episodes faster than TD3. Furthermore, TD3-FORK reaches higher maximum scores than TD3, and is more stable at these higher scores. Note that hyperparameters for TD3 were taken from [1] and hyperparameters for TD3-FORK were taken from [5]. Both algorithms did 12 episodes of random initial exploration, and decayed exploration noise once scores reached a threshold of 300+.

I also investigated the effect of adding the initial random exploration and decaying exploration noise on the convergence of TD3-FORK.

Figure 2b shows adding these features not only leads to faster convergence on high scores, but makes the agent considerably more stable once these high scores are reached.

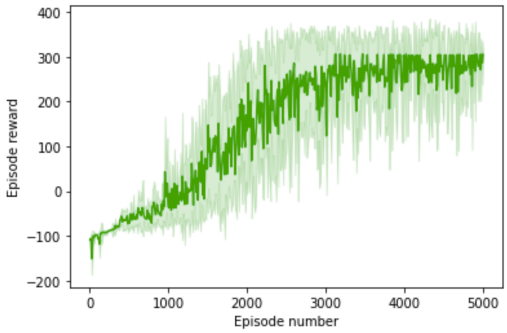

Figure 3: Convergence results for TD3-FORK on bipedal walker hardcore

TD3-FORK is also able to converge on bipedal walker hardcore with a slight change to the reward structure. I keep my implementation of TD3-FORK identical, but as suggested in [6] I change the -100 falling over reward to -5, and multiply all other rewards by 5. Furthermore, experiences from episodes where the walker doesn't fall are only added to the replay buffer 1/5th of the time, preventing the walker from further training on terrains it has already mastered. At around iteration 3000 the vast majority of runs score over 300, however the results are far less stable than on the basic bipedal walker. This is indicated by the highest average from 10 consecutive episodes being 307, but from 100 consecutive episodes being 293. However it is possible this could be improved with longer training.

The convergence results of TD3-FORK suggest that it is a good algorithm for tackling the bipedal walker environment, and is able to successfully make a bipedal robot walk consistently. Better and more stable convergence than the base TD3 algorithm also indicates that considering future states and values of these states is very effective in tackling tricky continuous environments. Experiments with adding initial random actions show that at the start of training increasing the amount of exploration can be very valuable, as it is only through exploration the walker can find an effective walking style. Only once an effective style is found and high scores are already being reached, is it worth exploiting high scoring strategies.

3 Limitations

Although TD3-FORK was able to successfully tackle the bipedal walker environment in less time and less episodes than TD3, running time was still an issue in the bipedal walker environment. When recording videos regularly it took towards 2 hours to run 700 episodes of TD3-FORK in the environment, meaning testing changes to algorithms and hyperparameters took a very long time. This means convergence results are shown from singular runs, and are not as reliable; results could have been considerably affected by random chance.

Future Work

In future work, running the algorithms used in this paper over multiple random seeds and averaging results would make sure the conclusions made in this paper are accurate. It would allow fairer comparison of algorithms and more concrete statements on the effectiveness of TD3-FORK as an extension to TD3.

Furthermore, with less time constraints the effectiveness of FORK applied to other popular policy gradient reinforcement learning algorithms like soft actor-critic could be investigated. The effectiveness of TD3-FORK on other environments could also be measured, to ensure the algorithm doesn't just improve performance on the environments seen in this paper. These experiments would show whether FORK is always worth adding to actor-critic algorithms in continuous environments, and if it should be a go-to method to solving many different reinforcement learning environments.

References

[1] Nikile Barhat. *TD3-BipedalWalker-v2-PyTorch*. URL: https://github.com/nikhilbarhate99/TD3-PyTorch-BipedalWalker-v2. (accessed: 12.24.2021).

[2] *BipedalWalker v2 · openai/gym Wiki*. URL: https://github.com/openai/gym. (accessed: 12.24.2021).

[3] 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.

[4] Timothy Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint* (2015). URL: https://arxiv.org/abs/1509.02971.

[5] Honghao Wei and Lei Ying. *FORK*. URL: https://github.com/honghaow/FORK. (accessed: 12.24.2021).

[6] Honghao Wei and Lei Ying. "FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning". In: *arXiv:2010.01652 [cs, stat]* (2021). URL: http://arxiv.org/abs/2010.01652.