Learning to Walk With TD3-FORK+BC

Anonymous author

Abstract

This paper aims to implement a TD3-FORK+BC agent on the BipedalWalker-v3 and BipedalWalkerHardcore-v3 environments provided by OpenAI Gym and explore agent performance in both.

1 Methodology

This paper proposes training a bipedal robot to learn to walk by sampling actions a from a modified TD3-FORK algorithm, which extends vanilla TD3.

The BipedalWalkerHardcore-v3 environment is a more difficult version of the BipedalWalker-v3 environment with ladders, stumps and pitfalls added in addition to the uneven terrain (Openai (2020)).

For both environments, a reward is given for moving forward. This reward totals to 300 points and over if the walker reaches the end of the terrain. A reward of -100 is given if the walker falls and the application of motor torque also results in negative reward (Openai (2020)).

The continuous action space of the Bipedal walker environments is shown in Table 1.

	Name	**Min**	**Max**
0	Hip 1 (Torque/velocity)	-1	$+1$
1	Knee 1 (Torque/velocity)	-1	$+1$
2	Hip 2 (Torque/velocity)	-1	$+1$
3	Knee 2 (Torque/velocity)	-1	$+1$

Table 1: The continuous action space of Bipedal Walker (Openai (2020))

Initially, the vanilla TD3 algorithm (Fujimoto, Hoof, et al. (2018)), a mode-free actor-critic technique, was implemented. The TD3 algorithm is an improved version of Deep Deterministic Policy Gradient (DDPG) (Byrne (2022)) and addresses some of its flaws such as reducing the overestimation bias which is caused by the algorithm consistently overestimating the Q values of the critic network. It does this in 3 ways (Byrne (2022)):

Twin Critic Networks. TD3 proposes the use of two critic networks to provide a clipped double Q learning technique. This uses the minimum value of the two networks when creating targets. By doing so, underestimation of Q values is favoured (Byrne (2022)) and target cannot introduce any further overestimation bias when compared to the standard Q-learning target (Fujimoto, Hoof, et al. (2018)). Thus a more stable approximation is provided. The target Q value update is calculated by:

$$y = r + \gamma \min_{i=1,2} Q_{\theta_{i'}}(s', \pi_{\phi'}(s')) \tag{1}$$

where r is the received reward, γ is the discount factor which determines the priority of short-term reward, s' is the next state, $Q_{\theta_{i'}}$ is the respective target Q network and $\pi_{\phi'}$ is the target actor network.

Delayed Policy Updates. To prevent the agent diverging by overestimating a bad policy (Byrne (2022)), TD3 proposes only updating the policy network every 2 time steps, unlike the value network which is updated each time step.

Target Policy Smoothing Regularization. When computing the targets, Gaussian noise ϵ is added and this is averaged over mini-batches (Byrne (2022)). Clipping in the range of a hyperparameter c is also applied to the noise to keep the target close to the original action. This technique makes the policy more stable as the target returns higher values for actions that are robust to noise (Byrne (2022)). This results in the target update being modified to the following:

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s^{'}, \pi_{\phi'}(s^{'}) + \epsilon), \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \tag{2}$$

The TD3 agent initially undergoes an observation stage in which actions are randomly sampled from the environment for a given number of time steps and the state, next state, action, reward, and done variable are added to the replay buffer. Building the replay buffer is essential as it allows the agent to sample from previous experiences when undergoing initial training.

After the observation stage, the agent is trained with a uniformly sampled mini-batch of transitions from the replay buffer, and the delayed updates to the actor and target critic network are applied as detailed above.

Following the completion of the observation stage, the agent selects an action with respect to its policy, as opposed to randomly sampling the environment.

The forward-looking actor (FORK) (Wei et al. (2020)) method was used to improve the performance of TD3, and the technique proposes the incorporation of two additional neural networks (Wei et al. (2020)) to TD3:

System network F_θ. This network predicts the next state given the current state and action:

$$\tilde{s}^{'} = F_\theta(s, a) \tag{3}$$

Smooth-L1 loss is used to train the system network after a mini-batch is sampled from the replay buffer:

$$L(\theta) = \|s' - F_\theta(s, a)\|_{\text{smooth L1}} \tag{4}$$

Modified Reward network R_η. This network predicts the reward given the current state, action and next state:

$$\tilde{r}^{'} = R_\eta(s, a, s^{'}) \tag{5}$$

MSE loss is used to train the network:

$$L(\eta) = \|r - R_\eta(s, a)\|^2 \tag{6}$$

The actor loss function $L(\phi)$ from TD3 is modified to use the system and reward networks to significantly improve performance. Although the forecasting element in the updated loss function provides a great improvement to performance, it's results are less desirable at the later stages of learning. To address lack of performance enhancement at the end of learning from the modified loss function, an adaptive weight (Wei et al. (2020))

$$w = 1 - clip(\left(\tfrac{\bar{r}}{r_0}\right), 0, 1)w_0 \tag{7}$$

was added to the loss function where $\bar{r}$ is the average cumulative reward per episode, r_0 is the predefined goal and w_0 is the base weight. This helps control the contribution of learning provided by FORK such that the learning is boosted by FORK at the beginning, but its contribution on the loss function gradually decrease it as training progresses.

Thus, the actor loss function combining all these elements can be written as:

$$L(\phi) = E[-Q_\theta(s, \pi_\phi(s)) - wR_\eta(s, \pi_\phi(s)) - w\gamma R_\eta(\tilde{s}^{'}, \pi_\phi(\tilde{s}^{'})) - w\gamma^2 Q_\theta(\tilde{s}^{''}, \pi_\phi(\tilde{s}^{''}))] \quad (8)$$

As suggested by Wei et al. (2020), rewards added to the replay buffer were increased by a factor of 5 and a reward of -100 was converted into -5 to help solve BipedalWalkerHardcore-v3.

Additionally, the authors believed that failed episodes provide a better learning opportunity than successful ones. To address this, failed episodes and successful episodes were added to the replay buffer with a 5:1 ratio where a failed episode is classed as those in which the bipedalwalker fell down (according to Wei et al. (2020)) and those where the episode reward is less than 250 (according to the implementation of Honghaow (2022)). Note that each episode is 2000 timesteps for this implementation.

In this project, further experimentation from those suggested by Wei et al. (2020) were also carried out as stated below:

Behaviour Cloning. The authors of TD3-BC (Fujimoto and Gu (2021)) propose a simple offline RL approach that involves the addition of a weighted behavior cloning (BC) loss to the policy update of TD3 to accelerate reinforcement learning from examples. This regularisation term was added to the TD3-FORK implementation, resulting in the following loss function:

$$L(\phi) = E[-\lambda Q_\theta(s, \pi_\phi(s)) - wR_\eta(s, \pi_\phi(s)) - w\gamma R_\eta(\tilde{s}^{'}, \pi_\phi(\tilde{s}^{'})) - w\gamma^2 Q_\theta(\tilde{s}^{''}, \pi_\phi(\tilde{s}^{''})) - \|\pi_\phi(s) - a\|^2] \quad (9)$$

This modification results in the proposed TD3-FORK+BC algorithm.

Hyperparamter testing. All the hyperparameters mentioned in Table 3 and Table 4 of Appendix A.4 in Wei et al. (2020) were used for the final implementation apart from the number of hidden units of the actor and critic networks. When experimenting with the number of hidden units, the parameters used in the original TD3 implementation proposed by Fujimoto, Hoof, et al. (2018) yielded better convergence results and were chosen over the 256 hidden units per layer for both actor and critic networks suggested by Wei et al. (2020).

Experimentation on the initial number of time steps allocated for the observation stage suggested that decreasing the number of time steps dedicated to random exploration to 5000 resulted in the agent being less stable and it was observed on multiple occasions that the episode reward drastically dropped to below 0 after appearing stable at 300+ points for several episodes. This could be a result of the agent not having enough previous experiences from random exploration, causing it to horribly fail when presented with a new challenge. Increasing this value to 20,000 resulted in the agent achieving rewards of 300 at much later episodes. Thus the number of time steps for random exploration was set to 10,000 as suggested by Wei et al. (2020).

Weight Initialisation. Additionally, although other authors like Rao et al. (2020) have suggested an Orthogonal initialisation scheme to be effective in initialising neural networks in a reinforcement learning (RL) setting to improve learning speed, experiments carried on this paper's implementation on the BipedalWalkerHardcore-v3 environment showed that a Xavier Uniform initialisation where values for each layer are sampled from

$$W \sim U\left(-\sqrt{\frac{6}{fan_{in} + fan_{out}}}, \sqrt{\frac{6}{fan_{in} + fan_{out}}}\right) \quad (10)$$

and bias set to 0.001 provided greater stability and faster convergence.

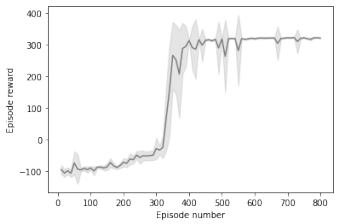

Figure 1: TD3-FORK+BC agent on basic environment for 800 episodes.

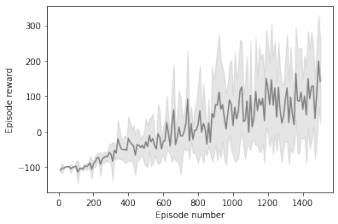

Figure 2: TD3-FORK agent on hardcore environment for 1500 episodes.

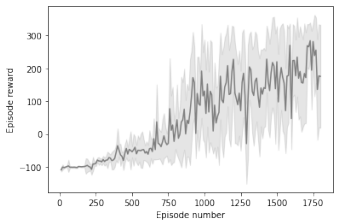

Figure 3: TD3-FORK+BC agent on hardcore environment for 1800 episodes.

2 Convergence Results

In the basic environment, the agent converges quickly and reaches a score of 300 between 300 and 400 episodes with little variability (see Figure 1) and achieves a reward of 300+ for over 100 consecutive episodes in the basic environment after episode 645. Although the agent was also seen reaching scores of 300 within 150 episodes, this observation was not recorded due to technical difficulties.

This contrasts the model's performance in the hardcore environment, where the episode rewards have a lot of variability. However, this is expected given the complex nature of the environment, and it should be noted that the rewards gradually increase as training progresses.

The addition of the BC term to the loss function of TD3-FORK provided significant performance improvement in the hardcore environment (see Figure 2 and 3), allowing the agent to consistently crawl over the bigger obstacles which was not observed through the basic TD-FORK implementation. Thus the agent is able to achieve scores of 300+ frequently.

However, it should be noted that on average, the TD3-FORK agent is able to achieve subtly higher speeds on the basic environment, and thus higher scores (observed up to 326) compared to the TD3-FORK+BC agent. This is because the TD-FORK agent unfailingly learns to run in a human-like manner, whereas the TD3-FORK+BC agent, on average, has a hybrid approach to walking that combines the TD3 action of propelling with the forward leg and the TD3-FORK running action. The agent performs best when it learns to utilise the human-like running motion more than the TD3 movement.

3 Limitations

Even after 1800 episodes of training in the hardcore environment, the agent does not consistently achieve scores of 300+. The positive trend of rewards suggests that more episodes of training are required.

Future Work

Future work could run the TD3-FORK+BC algorithm for a greater number of episodes on the hardcore environment. LSTM-TD3 (Meng et al. (2021)) is also an alternative to TD3 which suggests the use of recurrent LSTM layers as a replacement to some of the fully connected layers in both the actor are critic networks. Future studies should compare the performance of the LSTM-TD3 algorithm to that of TD3-FORK and TD3-FORK+BC and possibly look at updating both algorithms to utilise recurrent LSTM layers in the actor and critic networks to effectively extract experiences from history.

References

Byrne, Donal (July 2022). *TD3: Learning to run with ai*. URL: https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93.

Fujimoto, Scott and Shixiang Shane Gu (2021). *A Minimalist Approach to Offline Reinforcement Learning*. DOI: 10.48550/ARXIV.2106.06860. URL: https://arxiv.org/abs/2106.06860.

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). *Addressing Function Approximation Error in Actor-Critic Methods*. DOI: 10.48550/ARXIV.1802.09477. URL: https://arxiv.org/abs/1802.09477.

Honghaow, Wei (Apr. 2022). $Fork/TD3_FORK_BipedalWalkerHardcore_Colab.ipynbatmasterhonghaow/fork$. URL: https://github.com/honghaow/FORK/blob/master/BipedalWalkerHardcore/TD3_FORK_BipedalWalkerHardcore_Colab.ipynb.

Meng, Lingheng, Rob Gorbet, and Dana Kulic (2021). "Memory-based Deep Reinforcement Learning for POMDP". In: *CoRR* abs/2102.12344. arXiv: 2102.12344. URL: https://arxiv.org/abs/2102.12344.

Openai (2020). *Bipedalwalker v2 · openai/gym wiki*. URL: https://github.com/openai/gym/wiki/BipedalWalker-v2.

Rao, Nirnai et al. (2020). *How to Make Deep RL Work in Practice*. DOI: 10.48550/ARXIV.2010.13083. URL: https://arxiv.org/abs/2010.13083.

Wei, Honghao and Lei Ying (Oct. 2020). "FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning". In.