Learning to Walk Using TD3 and TD3-FORK

nvzf61

Abstract

This paper aims to learn to walk efficiently in the BipedalWalker-v3 environment using two related models: Twin Delayed DDPG (TD3) and a Forward Looking Actor for TD3 (TD3-FORK). We primarily improve perforance using a combined experience replay (CER) buffer and increasing exploration noise (with decay) alongside other small experiments and modifications. We also show our models ability to navigate more complex environments using BipedalWalkerHardcore-v3.

1 Methodology

1.1 Overview

Twin Delayed DDPG (TD3) [1] is an off-policy algorithm based on Deep Deterministic Policy Gradient (DDPG) [2] that addresses the issue of overestimation bias in DDPG using three tricks: Clipped Double-Q Learning, Delayed Policy Updates, and Target Policy Smoothing. TD3 is an actor-critic model such that the actor provides possible actions for the agent whilst the critic evaluates the given actions.

TD3-FORK [3] replaces the actor in TD3 with a forward looking actor that predicts future states and rewards to allow the agent to navigate more complex environments.

1.2 Background

Reinforcement learning aims to learn a reward-optimising policy for an agent in a given environment. We consider a standard reinforcement learning setting defined as a Markov decision process $(\mathcal{S}, \mathcal{A}, p_0, p, r, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the action space, $p_0(s_0)$ is the initial state distribution, $p(s_{t+1}|s_t, a_t)$ is the transition function, $r(s_t, a_t)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor.

At each discrete timestep t, with state $s_t \in \mathcal{S}$, the agent selects an action $a_t \in \mathcal{A}$ with respect to its policy $\pi(a_t|s_t)$, receiving a reward r_t and the new environment state s_{t+1} from the reward and transition functions respectively. To learn the optimal policy π_ϕ, we aim to maximise the expected return $J(\phi) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_0]$ with return $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$.

For simplicity let $s, a, r, s' = s_t, a_t, r_t, s_{t+1}$. In actor-critic methods, the policy (actor) is updated by taking the gradient of the expected return using the deterministic policy gradient $\nabla_\phi J(\phi) = \mathbb{E}_{s \sim p_\pi}[\nabla_a Q^\pi(s, a)|_{a=\pi(s)} \nabla_\phi \pi_\phi(s)]$ where $Q^\pi(s, a) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_t|s, a]$ is the value function (critic). For a large state space, we can estimate the value with a neural network $Q_\theta(s, a)$ with parameters θ. This network is updated using temporal difference learning with a secondary frozen target network $Q_{\theta'}(s, a)$ with objective value $y = r + \gamma Q_{\theta'_i}(s', \pi_\phi(s'))$.

1.3 TD3

Clipped Double-Q Learning. In DDPG, the critic can overestimate the value estimate causing the policy to break by exploiting this error - this is known as overestimation bias. To prevent it, we learn two critics $(Q_{\theta_1}, Q_{\theta_2})$ and update both using whichever target value is smaller to give the target update:

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

If $Q_{\theta_2} > Q_{\theta_1}$ then no bias has been induced; if $Q_{\theta_2} < Q_{\theta_1}$ then overestimation bias has occurred and the value is reduced to prevent it.

Delayed Policy Updates. High variance of the value estimate can contribute to overestimation bias and provides a noisy gradient for policy updates. Due to temporal difference updates, estimation error can accumulate over time. It is shown by Fujimoto et al. [1] that target networks can reduce the error over multiple updates whilst policy updates on high-error states cause divergent behaviour. As such, the policy and target networks are updated every two critic network updates to reduce variance and prevent divergent behaviour.

Target Policy Smoothing. In DDPG, the policy can overfit to sharp peaks in the value estimate. As such, we introduce a regulariser, by adding random noise to the target policy, to smooth out the value estimate. We clip the noise to keep the target close to the original action. This gives the modified target update including clipped double-Q learning:

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

1.4 TD3-FORK

We introduce two additional neural networks:

The System Network F_θ predicts the next state of the environment $\tilde{s}' = F_\theta(s, a)$. This network is trained with smooth-L1 loss $L(\theta) = ||s' - F_\theta(s, a)||_{smooth\ L1}$.

The Reward Network $R\eta$ predicts the reward $\tilde{r} = R_\eta(s, a, s')$. This network is trained with MSE loss $L(\eta) = ||s' - R_\eta(s, a, s')||^2$.

We can now introduce our forward looking actor (FORK) with the system and reward network as above. This new agent can forecast future states and rewards to improve its policy. We include an adaptive weight w in the loss function to accelerate learning at the start and decrease in weight towards the end to prevent errors/noise at the end of learning:

$$L(\phi) = \mathbb{E}[-Q_\theta(s, \pi_\phi(s)) - wR_\eta(s, \pi_\phi(s)) - w\gamma R_\eta(\tilde{s}', \pi_\phi(\tilde{s}')) - w\gamma^2 Q_\theta(\tilde{s}'', \pi_\phi(\tilde{s}''))] \tag{3}$$

$$where \quad \tilde{s}' = F_\theta(s, \pi_\phi(s)) \quad and \quad \tilde{s}'' = F_\theta(\tilde{s}', \pi_\phi(\tilde{s}')) \tag{4}$$

2 Experiments

2.1 BipedalWalker-v3

We first use the hyperparameters from the original TD3 implementation. We use the hyperparameters from the GitHub implementation as they provide performance improvements over those used in the paper.

Hyperparameter	**Value**
Buffer Size	1e6
Batch Size	256
Discount	0.99
Learning Rate	3e-4
Exploration Noise SD	0.1

Hyperparameter	**Value**
Tau	0.005
Policy Noise	0.2
Noise Clip	0.5
Policy Frequency	2
Start Timestep	25e3

Table 1: Hyperparameters for both TD3 and TD3-FORK

We then experiment with a prioritized experience replay (PER) [4] buffer before finding better performance with a simpler combined experience replay (CER) [5] buffer. A transition is defined as the tuple (s, a, r, s') and is added to the buffer at each timestep t. For our original replay buffer, we uniformly sample a batch of these transitions when training the agent. This allows our agent to reuse previous experiences to boost learning. For CER, when uniformly sampling a batch from the replay buffer, we sample one less and add the latest transition to the replay buffer instead. This effectively removes the need to optimise replay buffer size; all sizes will now result in a similar performance. CER also has the added benefit over PER of being $\mathcal{O}(1)$ rather than $\mathcal{O}(\log N)$ so is more computationally efficient.

We experiment with new hyperparameters with the focus of exploration/ exploitation optimisation. From our best combinations of hyperparameters we were occasionally able to achieve high scores in much fewer episodes, however the number of episodes was inconsistent and often worse than our initial hyperparameters. We also noticed large dips in reward even once our model had learned to walk. To mitigate this, we trialed various combinations of exploration noise decay factors and reward thresholds with varying success. Despite all these inconsistencies, we occasionally encounter a run that outperforms all our initial models. As such, we use an exploration noise of 0.3 with decay 0.995 at reward threshold 250 for our final model.

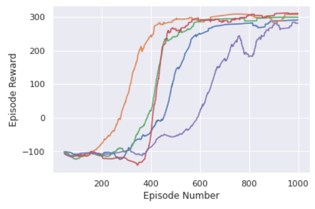
We also investigate Xavier weight initialisation for the actor and critic to maintain variance across all layers of the actor and critic networks, however we noticed no improvement.

2.2 BipedalWalkerHardcore-v3

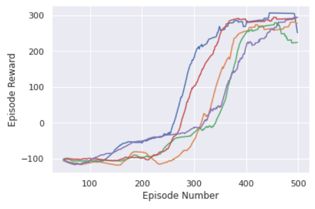
From Wei and Ying's implementation for solving BipedalWalkerHardcore-v3 [3], we adopt one of the two features added to aid solving the environment. Namely, reward scaling: the -100 reward (agent has fallen over) is changed to -5 and all other rewards are increased by a factor of 5.

3 Results

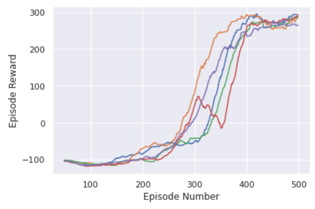
To test our models, we run 5 randoms trials on each model. We then take the average of these trials to give a reliable way to compare performance. We choose not to plot our results including their standard deviation as it would make the graphs much less intelligible. We also smooth our plots with a rolling mean of window size 50 to make them more comprehensible.



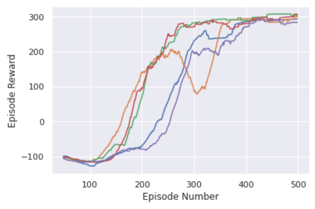
(a) TD3



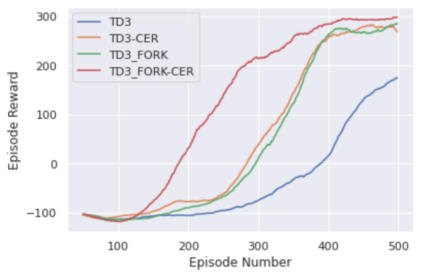
(b) TD3 with CER



(c) TD3-FORK



(d) TD3-FORK with CER



(e) Average of trials

Figure 1: Reward plots for 5 random trials and their average on given models

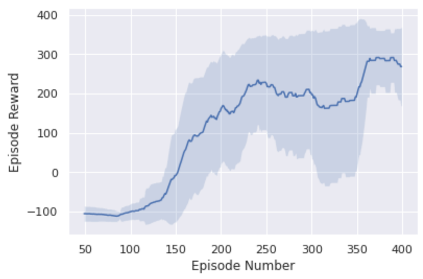
We only have a small sample size of 5 trials per model. As such, we have to be careful when drawing conclusions between models as these plots may not be fully representative of the performance of our models. For example, it is more than likely that we could still encounter convergence inconsistencies within our non-pure TD3 models, they just weren't present within our small set of trials.

We notice that pure TD3 suffers from convergence inconsistencies between trials; some trials take much longer to converge on an optimal policy than others. These inconsistencies are not as present in our other models. Despite this, all trials for TD3 fully converge in under 1000 episodes. All trials for the other models consistently converge in under 500 episodes.

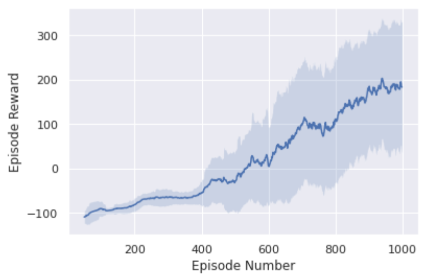
When comparing the average of all trials for each model, we can clearly see that TD3-FORK with CER is our best model and pure TD3 is our worst given how long they take to achieve high scores. Interestingly, TD3-FORK and TD3 with CER have very similar performance.

For our submission, we also include the reward plot from one trial (seed 0) of our final model for both BipedalWalker-v3 and BipedalWalkerHardcore-v3 to accompany the performance videos and log files. We smooth our results with a rolling mean of window size 50 and include a rolling standard deviation of window size 50.

As a reminder, our final model uses TD3-FORK with the original TD3 hyperparameters with exploration noise 0.3, decay 0.995, reward threshold 250, and a CER buffer.



(a) BipedalWalker



(b) BipedalWalkerHardcore

Figure 2: Reward plots for a single trial of final model with seed 0

Using the model described above, we can achieve scores above 250 in under 200 episodes for the BipedalWalker-v3 environment and above 250 in under 500 episodes for the BipedalWalkerHardcore-v3 environment.

4 Limitations

The main limitation associated with our final model is the high fluctuations in performance from increasing the standard noise deviation; some runs begin to converge very quickly whilst others take a lot longer. Similarly, we still observe occasional dips in reward, even with noise decay. However, the very small number of episodes required to achieve the high scores shown in our submission outweighs the occasional dip in performance that wouldn't be present if we were to revert to our initial hyperparameters.

Although Clipped Double-Q learning reduces overestimation bias, it may introduce underestimation bias, although it is preferable to the former as underestimated actions are not explicitly propagated through the policy updates.

5 Future Work

In the future, it would be beneficial to do more thorough hyperparameter optimisation.

We could also experiment with other variants of TD3-FORK presented by Wei and Ying. For example, FORK-DQ uses future action values instead of rewards in the loss function and was originally implemented by the authors to solve BipedalWalkerHardcore-v3. We could further combine these variants with more sophisticated replay buffers such as those provided by cpprb [6].

It may also be worth adopting the second change proposed by Wei and Ying to aid solving BipedalWalkerHardcore-v3 in fewer timesteps: failed episodes, where the agent fell down, and successful episodes are added to the replay buffer with a 5:1 ratio. This should increase performance as the successful episodes will no longer overwhelm the more useful failed episodes towards the end of training.

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: `1802.09477 [cs.AI]`.

[2] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: `1509.02971 [cs.LG]`.

[3] Honghao Wei and Lei Ying. *FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning*. 2021. arXiv: `2010.01652 [cs.LG]`.

[4] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: `1511.05952 [cs.LG]`.

[5] Shangtong Zhang and Richard S. Sutton. *A Deeper Look at Experience Replay*. 2018. arXiv: `1712.01275 [cs.LG]`.

[6] Hiroyuki Yamada. *cpprb*. Jan. 2019. URL: `https://gitlab.com/ymd_h/cpprb`.