Learning to Walk With TD3 and FORK

Anonymous author

Abstract

BipedalWalker is an OpenAI Gym environment which simulates a 2D robot walker. The goal of the environment is to learn how to efficiently walk a fixed distance. In the "Hardcore" extension of BipedalWalker the agent is faced with randomly generated states as well as different obstacles disrupting the agent's path. BipedalWalker and BipedalWalkerHardcore present various challenges for reinforcement learning algorithms. This paper proposes an ensemble approach combining the successes of TD3 and FORK into a single algorithm for sample efficient learning on the BipedalWalker-v3 and BipedalWalkerHardcore-v3.

1 Methodology

This paper proposes training a 2D bipedal robot to walk efficiently by using an ensemble of two previous approaches shown to be successful in continuous domains, namely TD3 and FORK [2], [8]. The Deep Deterministic Policy Gradient (DDPG) [6] algorithm is an actor-critic approach inspired by Q-Learning [7]. In deep reinforcement learning actor-critic approaches approximate the policy and value function with a neural network [5]. In this case, the Bellman equation describes an optimal action-value function $Q^*(s,a)$ as follows,

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

where $s' \sim P$ represents the next state s' being sampled from the environment's probability transition distribution $P(\cdot|s,a)$.

$$\phi_{target} \leftarrow p\phi_{targ} + (1-p)\phi, \tag{2}$$

where p is a hyperparameter between 0 and 1, usually near 1.

Similar to TD3, this paper uses two clipped Q-functions [2]. The target of the Q-functions is a single target defined as:

$$y(r,s',d) = r + \gamma(1-d)\min_{i=1,2} Q_{\phi_{i,targ}}(s', a'(s')) \tag{3}$$

Both critics are optimized with regression according to the equation below.

$$L(\phi_i, D) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right] \tag{4}$$

where $i \in [1,2]$ represents the respective critic.

Minimizing the Q-value for the target avoids overestimating the Q-value, which was a common problem of the DDPG algorithm that often causes the policy to collapse by diverging or getting stuck in local optima [2]. In this instance, $a'(s')$ represents the smoothed target policy. Actions for the Q-learning target are based on the policy μ_θ but with noise added. The result is then clipped into the action space range, as seen below.

$$a'(s') = clip(\mu_\theta(s') + clip(\epsilon, -0.5, 0.5), a_{Low}, a_{High}), \epsilon \sim \mathcal{N} \tag{5}$$

Target smoothing helps to regularize the algorithm, avoiding the issue DDPG faces in which an overestimate value update causes exploitation by the policy leading to the collapse of learning [2].

In actor-critic methods the policy, referred to as μ_θ is approximated with a neural network. In TD3, the policy is optimized by maximizing Q_{ϕ_1} according to the function below.

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D}[Q_{\phi_1}(s,\mu_\theta(s))] \tag{6}$$

This paper combines the methods of TD3 with a FORward looKing (FORK) actor method [8]. The method relies on an additional model, known as the system model, which learns to predict the next state by taking the current state and action as input. This model is used to modify the previously defined policy loss. The loss of the policy is defined according to the loss function below.

$$\begin{aligned} L(\phi) = \mathbb{E}[-Q_{\phi_{targ}}(s_t, \mu_{targ}(s_t)) - 0.5\lambda(Q_{\phi_{targ}}(\bar{s}_{t+1}, \mu_{targ}(\bar{s}_{t+1})) \\ - 0.25\lambda(Q_{\phi_{targ}}(\bar{s}_{t+2}, \mu_{targ}(\bar{s}_{t+2}))] \end{aligned}$$

where $\bar{s}_{t+1}$ and $\bar{s}_{t+2}$ represent the predicted state from the system model F_Ψ according to,

$$\bar{s}_{t+1} = F_\Psi(s_t, Q_{\phi_{targ}}(s_t)) \text{ and } \bar{s}_{t+2} = F_\Psi(s_{t+1}, Q_{\phi_{targ}}(s_{t+1})) \tag{7}$$

This allows the agent to consider actions two steps ahead of the current time step t. This is especially important for the Hardcore environment because policy updates will consider upcoming obstacles that are not encoding in the current state. The value λ represents the adaptive weight mechanism which makes the policy rely less on the future predicted states when the environment is close to converging. It is calculated according to the below equation,

$$\lambda = 1 - clip\left(\left(\frac{\bar{r}}{r_0}\right), 0, 1\right) \tag{8}$$

where $\bar{r}$ is the average cumulative reward and r_0 is a predefined goal in this case 300.

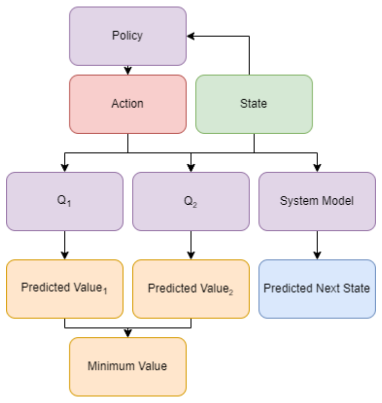

Figure 1: The policy network consists of 3 layers: 24x400, 400x300 and 300x4. The Q networks have 3 layers: 28x400, 400x300, 300x1. The system network has 3 layers: 28x400, 400x300, 300x24.

It should be noted the implementation uses a variation from the paper as the original paper uses an additional reward network to predict the reward for future states [4]. The policy loss only considers the system model losses if the system model loss is significantly small, otherwise it just relies on the objective from TD3 to update the policy. Figure 1 shows a simplified version of how data flows through the models.

The hidden layers of all networks use a Rectified Linear Unit (ReLU) activation function [3]. The policy function output layer uses a tanh activation function to squeeze the values into the range $[-1, 1]$ as required by the gym environment.

2 Convergence Results

The training was conducted with the same hyperparameters for both versions of the environment. The main difference was the update to data ratio, which reflects the number of updates to the network parameters with respect to the number of new interactions in the environment. Both environments used dynamically sized update ratios, in which a random minibatch of 100 is selected N times, where N represents the time steps of the most recent episode. The base environment update the networks $5 \times N$, with the policy target being updated every second update. The hardcore environment does the same but with a minibatch size of 200.

Figure 2 shows the mean episode reward for every 10 episodes when the model was trained on the simple environment.

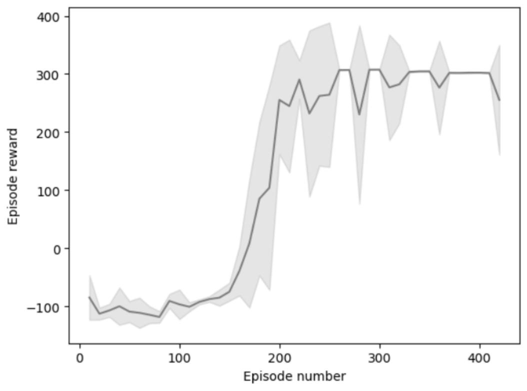

Figure 2: Mean episode reward every 10 episodes for the BipedalWalker environment. The gray bounds represent the standard deviation.

The model converges within 250 episodes, which is quicker than the basic TD3 implementation. This was achieved due to the FORK architecture as well as the higher update to data ratio. However, there are episodes where the model scores poorly, indicated by the variance in the mean and standard deviation of the plots in Figure 2. This suggests that the model needs more training data so that the policy stabilizes.

Figure 3 highlights the mean episode reward of the model trained on the hardcore environment. The model failed to converge in the given time-frame. It is clear from the plot that the model was learning faster than the original TD3-FORK paper, in which the model's average reward was approximately 0 by the 1000th episode [8]. This is likely a result of the increase in network updates per episode. Furthemore, the reward is still climbing which indicates that it will continue learning.

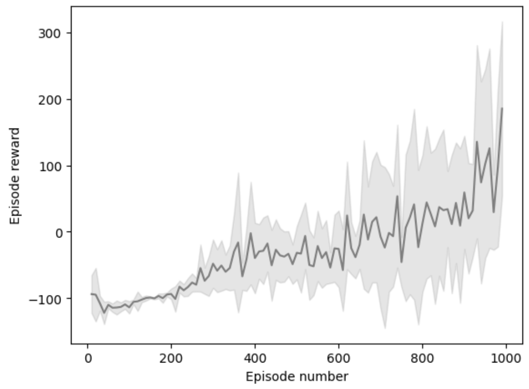

Figure 3: Mean episode reward every 10 episodes for the BipedalWalkerHardcore environment. The gray bounds represent the standard deviation.

3 Limitations

One limitation of the results presented is that the training efficiency is only evaluated on a single run of training. This means the effectiveness of the methods employed may be less considerable when averaged over numerous initial seeds. Furthermore, the robots performance could have been measured over multiple simulations with different seeds, which would better reflect the robots ability to walk, especially in the hardcore environments case as the levels are randomly generated. Similarly, the quality of the robots movement was not evaluated in the paper. For example, its ability to balance, walk intelligently and deal with obstacles.

The training efficiency of the model relied heavily on the update-to-data ratio, which allowed the network to learn more under the constraint of the episode limit. Although this method suggests that the model can learn from less training data, the trade-off is an increase in time of model updates. This means that the model is over-fitting to the collected experience instead of relying on new experiences. This could explain the results obtained in the hardcore environment which experiences more variance. It is likely that in more complex environments this method is less effective as over-fitting increases exploitation and reduces exploration.

Due to unexpected time constraints, the tuning of update-to-data ratio was limited. Although the training efficiency was improved, it is likely with finer tuning and more time to run the model it could have learnt more efficiently.

Future Work

TD3 aims to reduce variance in policy updates by maintaining two Q functions with a pessimistic bound on the expected reward to prevent the model overestimating the reward for a given state-action pair. In future an ensemble approach could have been considered that increased the number of Q networks and thus the effectiveness of training the policy [1]. Additionally, FORK relies on a system network to model the predicted next states. However, this approach only considers two future time steps relying on one model. As time steps in the continuous domain only represent a small portion of the future state, this approach could have benefited from using more models for future planning. As the hardcore environment requires adjusting movement to move over obstacles, more future planning would allow the agent to consider changes in action earlier. Moreover, the model should have been implemented with a separate model to predict rewards of future states instead of relying on a single Q function. It is likely that these methods would have catered to more reliable improvements to training efficiency.

References

[1] X. Chen et al. "Randomized ensembled double q-learning: Learning fast without a model". In: *arXiv* (2021).

[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] X. Glorot, A. Bordes, and Y. Bengio. "Deep sparse rectifier neural networks". In: *In Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), pp. 315–323.

[4] honghaow. *FORK*. `https://github.com/honghaow/FORK/`. 2020. URL: `https://github.com/honghaow/FORK/blob/master/BipedalWalkerHardcore/TD3_FORK_BipedalWalkerHardcore_Colab.ipynb`.

[5] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators." In: *Neural networks* (5 1989), pp. 359–366.

[6] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015). URL: `https://api.semanticscholar.org/CorpusID:16326763`.

[7] C.J.C.H. Watkins and P. Dayan. "Q-learning". In: *Machine Learning* (1992), pp. 279–292. URL: `https://doi.org/10.1007/BF00992698`.

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