Learning to Walk with Actor-Critic Methods

Will Farmer

Abstract

This paper demonstrates the drawbacks of actor-critic methods and how to improve their performance with forward-looking models. The potential of reward scaling and reward clipping are explored along with various ways scene information is recorded in replay buffers. This paper presents a TD3-FORK agent and compares its performance with that of a TD3 agent.

1 Methodology

This problem is decomposed into two distinct subtasks: learning to walk in a predictable, simple environment, and learning to walk in an unpredictable environment with obstacles and inclines. The predictable environment is known as the "easy environment" and the unpredictable environment is known as the "hardcore environment". The observation and action spaces for both environments are identical[1], with the action space consisting of 4 dimensions and the observation space 24 dimensions.

1.1 Learning to walk in the easy environment

Twin Delayed Deep Deterministic Policy Gradient (TD3) was selected as the Reinforcement Learning (RL) algorithm of choice. TD3 is an actor-critic method that has been shown to perform very well in continuous environments [2]. The method uses 2 critics and a single actor. The actor chooses what action $a \in A$ to take at each time step t based on the observation (state) $s \in S$ according to the policy $\pi : S \rightarrow A$, with the critic network critiquing the decision made at each timestep. Gaussian noise is then added to the action to encourage exploration of the environment to reduce the probability of the robot getting stuck at local maxima. Past actions, observations, and rewards are stored in a replay buffer after each timestep so that the critic and actor networks can be updated. The critic models are trained to minimise Mean Squared Error loss:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

where $\hat{y}_i$ is the model's predicted value and y_i is the ground truth. The policy $\pi : S \rightarrow A$ is updated every x timesteps, where x is known as the policy delay and is a value chosen by the programmer.

1.1.1 Hyperparameter selection

The hyperparameter values were initially chosen to be identical to the values in the reference code[2] to first ensure that the robot learned to walk. Once this was confirmed to be the case, this author used Optuna [1] to identify optimal hyperparameter values. The search space was initially defined to be large, so it could be narrowed over time to hone in on the optimal values for each hyperparameter. Each set of values would be run for 500 episodes. The highest episodic score, the mean episodic score, and the first episode to achieve a score greater than 300 are all recorded and used as success metrics. These metrics were chosen to identify whether the robot learned to walk at all, how quickly it learned to walk, and

[1] https://github.com/openai/gym/wiki/BipedalWalker-v2
[2] https://github.com/sfujim/TD3/blob/master/TD3.py

how consistently it was able to achieve high scores. The only parameter not tuned was the policy delay, which is recommended to always be every 2 timesteps. The suggested values in the reference code are provided along with the initial search spaces for each parameter in Table 1.

None of the first few iterations of hyperparameter tuning yielded results near the level of the suggested values. The collected data was still very useful, however, as this author was able to identify the parameter ranges that led to poor results and adjust the search space appropriately. While some positive results were obtained, the improvement in performance from the original parameters values was minimal.

1.2 Learning to Walk in the Hardcore Environment

When satisfied with the speed and sample efficiency of the agent in the easy environment, the agent was tested in the hardcore environment. Despite the success of the algorithm in the easy environment, the robot was unable to learn to walk in the hardcore environment, often getting stuck in a kneeling position or in the splits, likely to stop itself from falling over. The default environment reward for falling over is -100. This value aims to teach the robot that falling over is an undesirable characteristic. However, in the hardcore environment, falling over is a certainty in the early stages of training. These local maxima are extremely undesirable, as they also prevent the robot from exploring more of the action space. To combat this, this author implemented reward scaling, a technique that has been shown to occasionally improve results [5]. The reward for falling over was clipped to -20 and the reward for all other actions was multiplied by 5. This was done to still show that falling over was undesirable, but not so punishing that it would rather remain immobile instead. Although these changes yielded a small improvement and increased the amount of exploration, the model was still unable to walk.

1.2.1 TD3-FORK

The solution to this problem was the implementation of a forward-looking actor [6]. TD3-FORK was based on the original TD3 algorithm [2], but with the addition of a system model to predict the next state, reward scaling, reward clipping, and a change to the way the replay buffer stored past actions, observations, and rewards.

In the implementation of TD3 used in the easy environment, each observation, action and reward for each timestep was saved in the replay buffer. In TD3-FORK, the episodes in which the robot fell down or achieved a poor score and successful episodes are added to the buffer in a 5-to-1 ratio. The logic behind this being that the robot needs to learn how to keep walking in the different environments, and so it is more important for it to remember which actions led to poor results given its surroundings. This led the robot to fall over much more than before, but ultimately meant it learned to overcome different types of obstacle.

The reward for falling over was again clipped, but this time to a value of -5, hardly penalising the robot. All other actions were similarly multiplied by a factor of 5 to still demonstrate that falling over was undesirable while encouraging exploration. To further encourage exploration of the environment, the actions made in the first 10,000 timesteps are all made randomly and not according to the policy.

The greatest difference between TD3 and TD3-FORK was the implementation of the system network. This network was designed to predict the next state s_{t+1} from any given state s_t and was trained on MSE loss (equation 1). The system loss is then also used to calculate the actor loss for policy updates. This is ultimately what led the robot to learn how to walk in the environment, as the agent was no longer having to rely only on past experiences. Combined with the new format of the replay buffer, the robot can see what made it fall over last time, simulate what would happen if a different action was taken, and then select a new action. If this new action still results in an undesirable outcome, the system model is updated to improve accuracy.

Another difference, albeit a small one, is that if the hardcore model takes the mean score of the last 100 episodes into account when calculating the actor loss. This is done to slow learning when approaching convergence.

1.2.2 Hyperparameter Selection

Again, the same parameters suggested in the reference code[3] were initially used to ensure that the robot would learn to walk. Once this was confirmed, this author iteratively selected new values for the parameters and ran them for 2500 episodes, collecting the highest score achieved in an episode, the first episode to achieve a score greater than 300, and the mean score achieved over all 2500 episodes. Rather than using Optuna to search the entire space, this author opted for a Greedy approach instead. There are simply too many parameters in TD3-FORK to efficiently tune them all in conjunction with each other. Hence, this author first tuned the reward clip from falling over, then the value all other actions were multiplied by. In this instance, the learning rate of each of the models, exploration and policy noise and clips, discount, and soft update factor were left untouched from their suggested values. The reward for falling over was changed from -100 to -10 after searching through all integers i, $-20 \leq i \leq 0$. After searching through all integers i, $1 \leq i \leq 10$ for the value of the multiplier applied to the reward for all other actions, the integer 5 was still shown to perform the best. To check if the best value was in fact not an integer, all values $5 + \frac{n}{10}$, where $-5 \leq n \leq 5, n \in \mathbb{Z}$. Despite this iterative search, the value of 5 was still determined to be the best value and was hence selected for use. Given that no improvement could be found by tweaking the reward multiplier, this author decided to also tune the batch size. The recommended size was 100 samples, much smaller than the recommended batch size used in the vanilla TD3 algorithm (256). Using Optuna to suggest random integers in the range $[64, 256]$, this author observed scores for episodes as high as 327! However, the mean scores for these models after 2500 episodes were notably lower than that with a batch size of 100 and also took an additional 200 episodes to first score over 300.

1.2.3 Testing the TD3-FORK model in the easy environment

Given that the TD3-FORK model performed dramatically better in the hardcore environment than the TD3 model, it seemed logical to test the hardcore model's performance in the easy environment. The improvement was astonishing and hence the TD3-FORK model was chosen to perform as the agent for easy and hardcore environments. The model hyperparameters are different for each environment as a result of further tuning on the easy environment.

2 Convergence Results

2.1 Easy Environment

The agent consistently first navigates the whole environment after 126 episodes and converges after 208 episodes. The implementation is robust and completely reproducible, while also always achieving a high score of 327.6 and a mean episodic score of 291.4 over 2000 episodes. For comparison, the TD3 model would first score over 300 after 260 episodes and would not converge until after 800 episodes! Figure 1 shows the difference in performance between the two models.

2.2 Hardcore Environment

The model consistently achieves a score greater than 300 after 245 episodes and achieves a high score of 323.0. After only 2500 episodes, the mean score for all episodes is already 152.8, an extremely high average given the number of times the robot falls over at the start of training. The model is extremely robust and reliable. In five repeated simulations of 2500 episodes in the hardcore environment, the mean score, the highest score attained, and the first episode to reach a score greater than 300 were identical each time.

[3] https://github.com/honghaow/FORK/blob/master/BipedalWalkerHardcore/TD3_FORK_BipedalWalkerHardcore_Colab

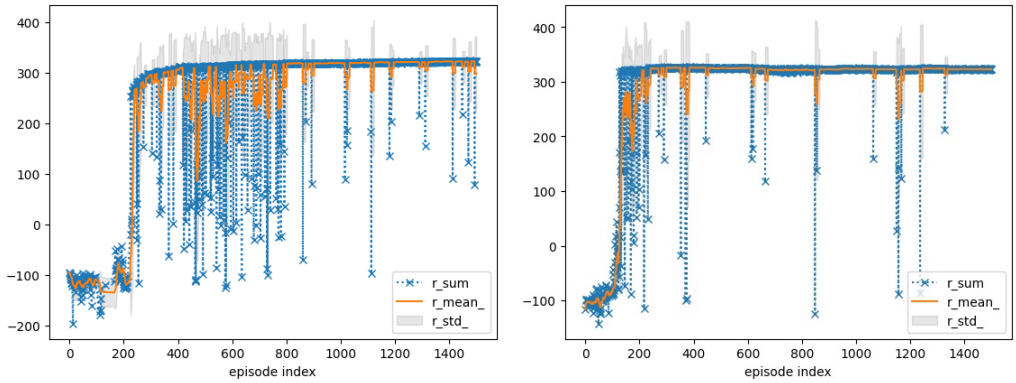

Figure 1: The scores achieved during training of the model on the easy environment when using TD3 (left) and TD3-FORK (right).

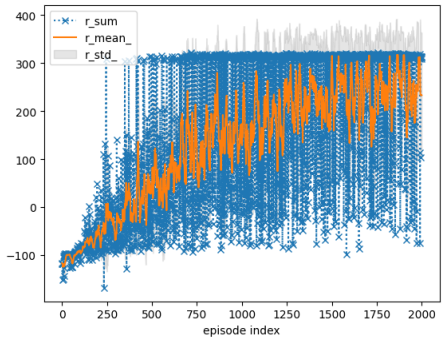

Figure 2: The scores achieved by the tuned TD3-FORK model on the hardcore environment.

3 Limitations

While the agent is able to consistently obtain high scores, the nature of the replay buffer means that even after convergence it still sporadically obtains very low scores in both environments. Additionally, while it first fully traverses the terrain after only 245 episodes in the hardcore environment, it takes thousands more episodes to converge. This implies that the model is not very sample efficient when it comes to identifying how to overcome specific obstacles.

Additionally, the performance of the agent in the hardcore environment could be improved further by tuning the parameters in parallel rather than greedily tuning one at a time, though this would be a more time-consuming process.

It is also possible that the agent performs worse on different seeds for both environments. The agent had its hyperparameters tuned on the same seed of both environments. Hence, it is possible that they have been overfitted to their environments and have lost generalisability.

Future Work

In the future this author would like to attempt to use RL methods that have not yet been shown to achieve high scores or converge in the bipedal walker environment. This would involve more experimentation and tuning compared to the hyperparameter tuning and reward scaling shown in this paper. Methods such as Temporal Difference (TD) Learning have been shown to extract useful features from various problem spaces [3][4]. TD learning could be applied to the bipedal walker environment to extract useful features from the observation space and help predict the consequences of future actions.

References

[1] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[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] Richard S. Sutton. "Learning to predict by the method of temporal difference". In: *Machine Learning* 3.1 (1988), pp. 9–44. URL: https://doi.org/10.1007/BF00115009.

[4] Gerald Tesauro. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68. URL: https://doi.org/10.1145/203330.203343.

[5] Tingwu Wang et al. *Benchmarking Model-Based Reinforcement Learning*. 2019. arXiv: 1907.02057 [cs.LG]. URL: https://arxiv.org/abs/1907.02057.

[6] Honghao Wei and Lei Ying. *FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning*. 2021. arXiv: 2010.01652 [cs.LG]. URL: https://arxiv.org/abs/2010.01652.

A Appendix

Table 1: The Initial Hyperparameter Search Spaces for the TD3 Model

Parameter	Suggested Value	Search Range
Learning Rate	0.0003	$[0.00001, 0.001]$
Batch Size	256	$[64, 1024]$
Discount	0.99	$[0.95, 0.9999]$
Soft Update Factor	0.005	$[0.001, 0.02]$
Policy Noise	0.2	$[0.1, 0.9]$
Noise Clip	1	$[0.5, 1.5]$
Exploration Noise	0.2	$[0.1, 0.9]$
Exploration Clip	0.5	$[0.1, 0.9]$