Learning to Walk with Enhanced TD3

Timofei Zavileiskii

Abstract

This paper explores enhancements of the TD3 algorithm for Bipedal-Walker environment to ensure faster learning and convergence. The improvements included, adding a probability distribution when choosing replays from the replay buffer, a few first episodes included random initialisation, altering the reward function, and adding scheduling for the hyperparameters. In the end, agent was able to get consistent scores over 300 after 80 episodes, and for hardcore environment, 10 episode averages of 200 after 1000 episodes.

1 Methodology

A bipedal robot is taught to walk on the terrain with an iterative approach. An already existing solution is taken as the baseline and improved step by step by adding algorithmic enhancements to improve speed of learning and convergence. First, the priority was given to ensure that agent learns optimal policy quickly and then scheduling was added to accelerate convergence.

1.1 Environment

The paper will develop agents for the 2 environments provided by the OpenAI Gym [4]. It requires for a bipedal robot to walk in a 2D environment, powered by box2D physics engine. The first normal environment, has mostly even terrain, while the hardcore version has obstacles scattered randomly.

The observation space has 24 dimensions and action space has dimension 4. The observation space consists of the hull angle speed, horizontal speed, vertical speed, angular velocity, joint positions, and joints' angular speed, whether legs touch the ground, as well as 10 lidar readings under the robot, giving distance to the nearest point on the ground. The action space is the torque for each of the 4 servos in robot's legs.

Positive reward is given for moving forward, and some negative reward is given for applying torque on servos and -100 when robot's hull touches the ground, ending the episode.

It is important to note that actor does not have perfect information as exact environment and agent's coordinates are not given in an observation.

1.2 Summary of Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique which allows to train an agent to find an optimal policy by trial-and-error by interacting with an environment. Classical RL is usually modelled with Markov decision process on Markov chains (set of states and transitions, where probability of a transition depends only on the current state), and where value of a state and action are defined with Bellman's Equations:

$$V(s) = \sum_a P(a|s)Q(a,s) \qquad Q(s,a) = R(a) + \gamma \sum_{s'} P(s'|a)V(s')$$

Where V(s) is value function for a state, Q(s, a) is value for the action a in the state s, and γ is discount for future rewards.

The major development for finding optimal policy is Temporal Difference algorithm [13]. It allows to evaluate the approximate reward of a transition using a recursive formula, which became the basis for many RL methods in the future.

$V(S_t) \leftarrow V(S_t) + \alpha(\gamma V(S_{t+1}) + R_{t+1} - V(S_t))$ $\delta_{t+1} = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

Where R_{t+1} is reward at time t+1, and δ_t is error for time step t, and α is a constant parameter.

Q-learning allows to approximate value function for an action in a similar way [15], and define an off-policy greedy learning policy.

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$

1.3 Starting Point - TD3

The starting point of the implementation was taken from the public GitHub repository by Momin Haider [1] and followed the TD3 algorithm [5], developed in 2018.

Twin-Delayed DDPG (TD3) architecture is an improvement upon the DDPG algorithm [11], which uses Actor Critique architecture to extend Q-learning to continuous action and state spaces. The core idea of DDPG is to use 2 models, with one approximating the value function of an action (Critic) and the other one trying to choose an action which maximises critic's estimation (Actor). Every time step, state, next state, performed action and reward are saved into the replay buffer, which is used for model training between steps. Training is done according to Temporal Difference with the updates using the difference with soft target as the value for the error. Exploration noise $\mathcal{N}(0, \sigma_a)$ is added to actions in the environment. Parameter τ is used to slow down updating of the target network, taking weighted average between its parameters θ_{target} and parameters of the trained network θ: $\theta_{\text{target}} \leftarrow \tau * \theta_{\text{target}} + \theta(1-\tau)$.

The TD3 improves this approach in 3 ways. First, it adds Gaussian noise (noise with normal distribution $\mathcal{N}(0, \sigma_t)$) to target action during training to reduce exploitation of the errors in the value function. Second, it uses 2 critique networks, and take the minimum when evaluating an action. Third, it trains the actor and update the policy every 2 steps, while training critic networks every time.

In this case, every model is a model has 4 layers, with ReLU activation function $f(x) = \max(x, 0)$ in the middle, first introduced in [6]. Critics have no activation function for output, and agent uses $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ to bring output into the -1, 1 range (fig 1).

Adam optimiser [9] is used as it tends to be less sensitive to hyperparameters, helping to reduce amount of hyperparameters to tune.

The initial hyperparameters were set to:

learning rate	τ	γ	σ_a	σ_t	batch size
0.0003	0.05	0.99	0.1	0.2	100

training clip	policy delay	buffer size
0.5	2	1,000,000

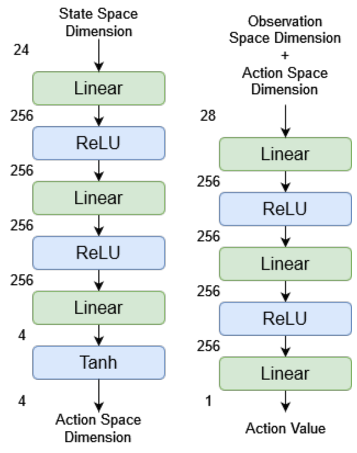

Figure 1: Actor Network (Left) and Critic Network (Right)

1.4 Initial Tuning

Figure 2: Robot stands

The initial performance was very mediocre with the model converging at approximately 600 steps in the normal environment (fig 3).

The first thing done was to try to improve the hyperparameters. We used Optuna framework [3] with 10 trials, setting the learning rate from 0.0001 to 0.005 and tau values from 0.01 to 1. The function to optimise was average for the last 200 episodes of 400 run to find parameters for which the agent would have converged by that point. The new found parameters were τ: 0.25 lr: 0.00526, giving convergence at 200 steps (fig 4).

Another small improvement was taken from another repository which solves Hardcore [8] environment by lowering the minimum reward of the episode to -5. In this case, minimum reward of -10 was used for normal environment. It helped to encourage exploration by reducing the error for falling down to prevent the agent from getting stuck in local minimums, as otherwise it often stays still to avoid the fall punishment (fig 2) for many episodes. It improved getting 300 average to roughly 140 steps (fig 5).

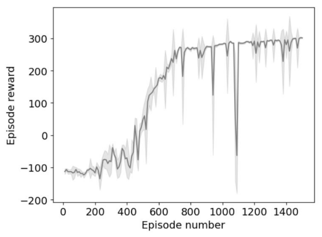

Figure 3: Initial Performance

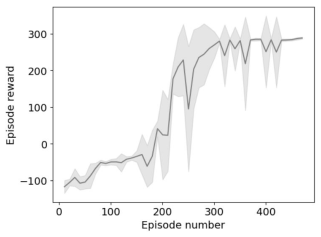

Figure 4: Performance After Hyperparameter Tuning

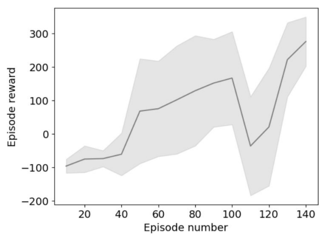

Figure 5: Performance After Introduction of Minimal Reward

1.5 Architecture and Training Improvements

Further improvements were taken from another repository with agent to solve Gymnasium environments, including the bipedal walker [2], which also provided a short book with description of used techniques [14].

This paper's implementation borrowed 2 improvements to improve the training and make sure that agent finds optimal policy within 100 episodes. The first improvement was to add a probability distribution to the sampling from the random buffer, which depends on fade factor set to 7. Every index i is assigned weight W_i and then probability P_i is obtained by normalising the weights to make sure cumulative probability 1.

$W_i = 10^{-7} * \tanh(\text{fade\_param} * (\frac{i}{\text{buffer\_size}})^2) \quad P_i = \frac{W_i}{\sum_i W_i}$

The second improvement was to reinitialise the agent with Xavier initialisation for weights, and setting biases to 0, first 5 episodes to get initial data for replay buffer.

1.6 Scheduling

To balance exploration and exploitation, Cosine and Linear scheduling were tried for hyperparameters. Scheduling is used on **learning rate**, τ, σ_t, σ_a, **min reward**, and **batch size**. For step t, total steps t', min value $x_{\min}$, and max $x_{\max}$, cosine scheduling is: $x_{\min} + 0.5(x_{\max} - x_{\min})(1 + \cos(\frac{t}{t'}\pi))$ and linear scheduling is: $x_{\min} + (x_{\max} - x_{\min})(1 - \frac{t}{t'})$.

In the end, linear schedule was used for most variables, while **learning rate** used cosine in both environments and **batch size** used cosine in normal. For the 2 parameters, t' was set to be twice as for the rest of parameters.

The scheduling was set to be conditional on the first decent attempt, when the final amended reward (with capped minimum) would exceed threshold **stall reward** as it was noticed that agents tends to converge shortly after making first few steps.

2 Convergence Results

Gamma, update delay, training clip were unchanged. In the end, normal environment used uniform sampling from the replay buffer, while hardcore used the probability distribution from section 1.3.

The parameters for normal were chosen to prioritise fast exploration and fast convergence with the smallest number of episodes. The parameters for hardcore were to ensure long and stable exploration as the challenge was to be able to find a consistent policy at all.

	learning rate	learning rate$_{\min}$	τ	$\tau_{\min}$	σ_a	$\sigma_{a\ \min}$	σ_t	$\sigma_{t\ \min}$
Normal	0.000526	0.00001	0.25	0.05	0.18	0.05	0.2	0.2
Hardcore	0.000526	0.00005	0.25	0.05	0.18	0.1	0.2	0.125

batch size	batch size$_{\min}$	min reward	min reward$_{\min}$	t'	stall reward	buffer size
100	512	-8	-100	60	60	1,500,000
100	512	-8	-30	800	280	2,000,000

In normal environment, the agent finds a good policy in in around 60 episodes, and gets scores above 300 regularly after 80 episodes. The main issue is lack of stability as after hundreds of episodes, at around 600, the agent will temporarily get stuck in a local minimum. For hardcore, the agent starts to reach finish after 500 episodes and receives average reward of 200 over 10 episodes after 1000 episode training.

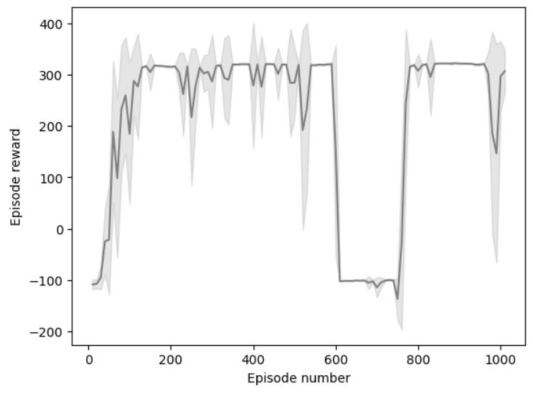

Figure 6: Normal Environment Performance

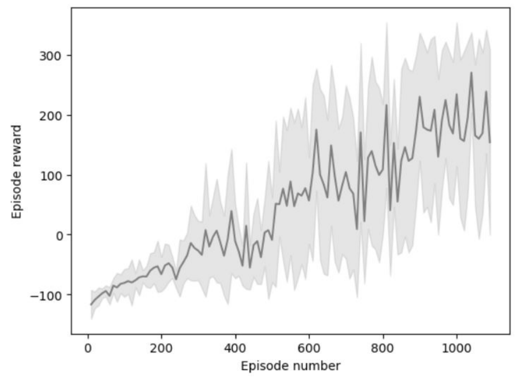

Figure 7: Hardcore Environment Performance

3 Limitations

The agent does show decent convergence results, however they can be improved further as seen on the OpenAI gym leader board with the best solution, claiming to solve it in 28 episodes [10]. The main issue, the lack of stability, was likely due to agent eventually learning to exploit errors in both critics, resulting in terrible scores (fig 7) for more than 100 episodes. It is unclear how to completely eliminate such possibility at all episodes, without completely stopping the agent learning for this model.

The hardcore environment was still unsolved with the agent frequently falling and receiving the fall punishment. It is likely a more complex architecture or way more training would be required to fully converge.

Future Work

Due to poor performance of the introduced probability distribution (not being used on normal), it would be great to consider other non-uniform ways to pick replays for training. For example, replays can be picked based on the δ-error [12], representing how novel was the experience. It might also help to get out of local minimums quicker.

Another improvement would be to introduce world-models as they are generally outperform model-free methods in terms of number of episodes required for convergence. An improvement of TD3 would be TD3 Forward Looking Actor (TD3-FORK) [16]. It learnt world model to improve value estimation by using short-term predictions of the world model.

Another For example, the Dreamer architecture can be tried, which was introduced by Google in 2019 for continuous control tasks [7]. It trains a world model on the experiences from the environment, while the action model learns in the latent space on trajectories generated by the world model.

References

[1] Implementation of TD3 Reinforcement Learning Architecture. URL: `https://github.com/hmomin/TD3-Bipedal-Walker`.

[2] Simphony network. URL: `https://github.com/timurgepard/Simphony`.

[3] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[4] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016). URL: `https://arxiv.org/abs/1606.01540`.

[5] 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.

[6] Kunihiko Fukushima. "Cognitron: A self-organizing multilayered neural network". In: *Biological cybernetics* 20.3-4 (1975), pp. 121–136.

[7] Danijar Hafner et al. "Dream to control: Learning behaviors by latent imagination". In: *arXiv preprint arXiv:1912.01603* (2019).

[8] TD3 Implementation which. URL: `https://github.com/ugurcanozalp/td3-sac-bipedal-walker-hardcore-v3`.

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] Leader board for OpenAI gym environments. URL: `https://github.com/openai/gym/wiki/Leaderboard`.

[11] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[12] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[13] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3 (1988), pp. 9–44.

[14] Michele Folgheraiter Timur Ishuov. *How to make Robot move like a Human with Reinforcement Learning*. Ed. by Madi Nurmanov Abdubakir Qo'shboqov Zhenis Otarbay. 2023.

[15] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[16] Honghao Wei and Lei Ying. "Fork: A forward-looking actor for model-free reinforcement learning". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 1554–1559.