Learning to Walk with TD3

Abstract

Through the use of twin delayed DDPG (TD3), this paper outlines the method and results of training a bipedal robot, with four available actions, to walk quickly by learning with the least amount of interaction within the OpenAI BipedalWalker-v3 environment. This includes an explanation of the actor-critic architecture for TD3 and the implementation of this model. In addition to an analysis of the results and quick convergence of the model, when trained within this environment, and the lack of convergence in the hardcore version. And finally a discussion of issues faced and further enhancements that could be made, such as the use of a forward looking actor.

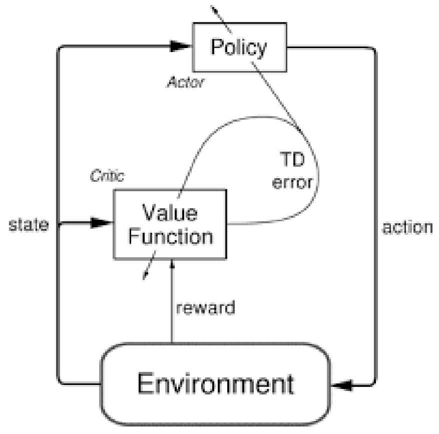
1 Methodology

The aim of this paper is to train a bipedel robot to walk quickly, with the least interaction within a 2D environment. The algorithm proposed is twin delayed DDPG (TD3), which is a popular, recent deep reinforcement learning algorithm, which uses a model-free reinforcement learning approach. This involves an agent that requires trial-and-error experience to find the optimal policy within an environment.

The environment set up was OpenAI's BipedalWalker-v3. The action set includes the movement of a hip and knee on each side of the agent's body. To solve the environment, a total reward of over 300 is required, with an added reward of -100 if the agent's head hits the ground.

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \text{ [12]} \tag{1}$$

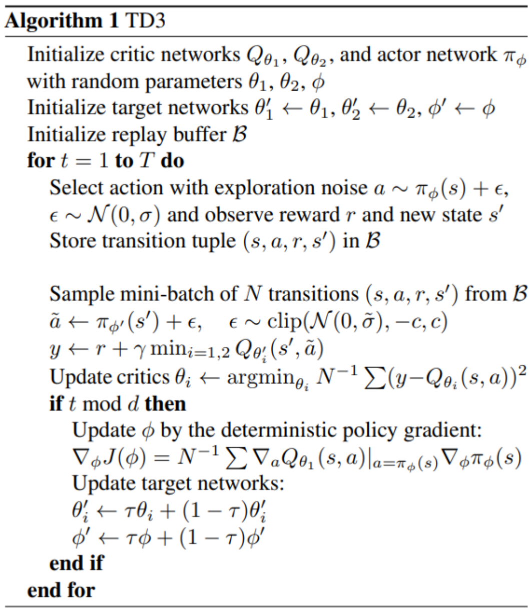
TD3 is a hybrid model, which combines Q-learning and policy gradients to map the states to actions, to learn this map and the value of actions in those particular states. The reward function is defined as seen in (1), which considers the Markov chain given a stochastic policy, the actions given a set of states and the value of the action state pairs.



[9]

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2}$$

To optimise the policy, an actor-critic model structure for a continuous action space is used by TD3, as seen in the figure above. A critic adjusts the parameters of the value function (through the TD error (2)), whereas an actor adjusts the policy parameters (θ) as suggested by the critic. In this instance, the actor creates action probabilities between -1 and 1 after taking in 24 parameters. The TD error is simply the difference between the current value estimate, the discounted value estimate and the reward gained from transitioning [5]. This is a form of temporal difference (TD) learning, as policies are modelled independent to the value function.



```
**Algorithm 1** TD3
  Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$
  with random parameters $\theta_1$, $\theta_2$, $\phi$
  Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
  Initialize replay buffer $\mathcal{B}$
  for $t = 1$ to $T$ do
    Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
    $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
    Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    if $t$ mod $d$ then
      Update $\phi$ by the deterministic policy gradient:
      $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
      Update target networks:
      $\theta'_i \leftarrow \tau \theta_i + (1 - \tau)\theta'_i$
      $\phi' \leftarrow \tau \phi + (1 - \tau)\phi'$
    end if
  end for
```

[3]

The hybrid DDPG algorithm policy model is extended by TD3, to prevent the issue of overestimation [3] with the value function through the Q-learning algorithm. This is completed through clipped double Q-learning, which uses the minimum estimation to favour underestimation when learning the value of an action in a state. Additionally, TD3 updates the policy less frequently than the value function, to prevent the value being overestimated when the policy is weak, and vice versa. Finally, target policy smoothing is implemented to avoid overfitting the value function, hence clipped random noise is added and averaged over batches.

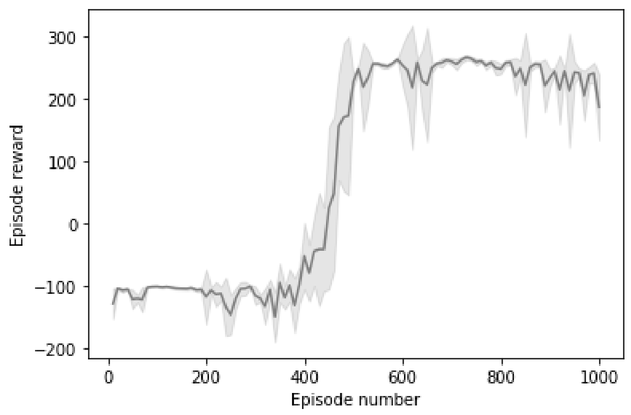
Due to these strengths and enhancements, TD3 was chosen to be implemented. When compared to other algorithms, such as Proximal Policy Optimization (PPO) which has a higher training speed, TD3 is seen to perform better on average among a variety of different domains [7].

The pseudo-code implementation above, is seen in the original paper [3]. This TD3 model was implemented using previous coding examples such as [6, 4] and the model which came with the original paper [2] and to ensure a robust implementation.

Firstly, the hyper-parameters were tuned to those in the original TD3 paper. The actor-critic architecture was then initialised, including critic Q1 and Q2 networks and an actor network. An experience replay was then implemented for when a policy is executed. Finally, the agent was defined, which includes the actions it could select, which adds exploration noise discussed to the environment, as discussed previously. In addition to the process of training, which includes sampling a batch of the transitions from the experience replay to update the target networks, selecting an action and then applying target policy smoothing, calculating critic Q values, calculating the loss for the two critic networks and updating the actor, actor loss and target networks. Save and load states were also implemented to allow agents to be reloaded to continue training.

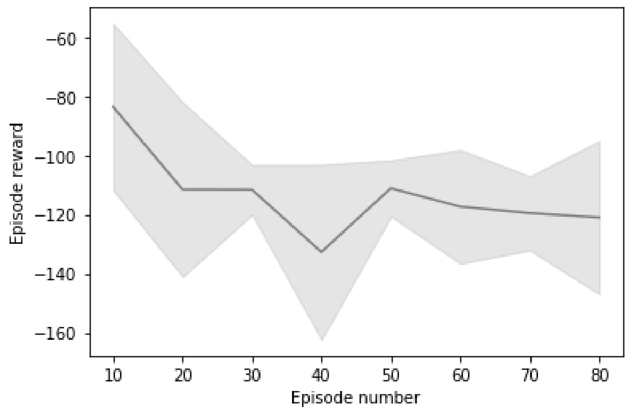
This agent was trained for 1000 episodes (11.5 hours), with video recordings taken every 50 episodes. This was then trained for 250 episodes in hardcore mode and 250 episodes in hardcore mode using the pre-trained agent from normal mode.

2 Convergence Results



As seen in the image above, the model converges at a positive score between 240 and 260 - around 550 episodes. The highest score achieved was a reward of 271.6 at episode 725.

After the high score is attained, the reward steadily declines at a very slow rate. Although the agent does not reach the optimal reward, it consistent attains very high results after 500 episodes. Moreover, it is eventually able to learn the optimal policy of balancing it's knees, and is able to walk fluently and fast, navigating the terrain well. The agent stops falling around 400 episodes and scores above zero close to 500 episodes. There is a drastic change in the average reward between 450 and 550 episodes.



When the algorithm was run in hardcore mode, the reward stayed roughly the same around -110, as seen in the hardcore mode log, and did not converge. In most cases, the agent was stuck in a splits position (and so also took a long time to train). When the pre-trained agent was plugged into the hardcore mode environment, the model performed even more poorly, as seen in the figure above.

3 Limitations

In regard to the training of the model, an error was found with the save function, where the critic was overwriting the actor. Hence, an agent was trained for 10 hours on two separate occasions, before crashing, and a reward plot was not saved. This error was later corrected, along with the addition of a Google drive mount, to properly save logs and videos.

In addition, more episodes could have been run to see if there were any improvements, or a better algorithm could have been implemented to solve the environment (consistently achieve a reward of 300) and also solve the hardcore mode environment.

Future Work

The TD3 algorithm could be further extended to achieve this goal. One option is to implement a forward looking actor [11, 10], to understand the terrain coming next and what action should then be taken, such as our approach in real life. Another option is combining TD3 with behavioural cloning, to help the agent pick actions closer to that which a real human would choose [1]. Instead of TD3, an evolving stable strategy could be implemented, such as OpenAI-ES, where gradients do not need to be fully calculated [8].

Additionally, an adjustment that could have been made to this TD3 implementation was to use a smaller alpha value - as temporal difference learning is unstable - to prevent reconvergence to the optimal solution. It is likely that a little after 1000 episodes, the average reward would have reduced a fair amount.

References

[1] A. Beeson and G. Montana. *Improving TD3-BC: Relaxed Policy Constraint for Offline Learning and Stable Online Fine-Tuning.* `https://arxiv.org/abs/2211.11802`. [Online; accessed 09-Feb-2023]. 2022.

[2] S. Fujimoto. *TD3.* `https://github.com/sfujim/TD3/blob/master/TD3.py`. [Online; accessed 09-Feb-2023]. 2018.

[3] S. Fujimoto, H. van Hoof, and D. Meger. *Addressing Function Approximation Error in Actor-Critic Methods.* `https://arxiv.org/pdf/1802.09477v3.pdf`. [Online; accessed 09-Feb-2023]. 2018.

[4] Higgsfield. *BipedelWalker.* `https://github.com/higgsfield/RL-Adventure-2/blob/master/6.td3.ipynb`. [Online; accessed 09-Feb-2023]. 2018.

[5] J. Hood. *Reinforcement Learning: Temporal Difference (TD) Learning.* `https://www.lancaster.ac.uk/stor-i-student-sites/jordan-j-hood/2021/04/12/reinforcement-learning-temporal-difference-td-learning`. [Online; accessed 09-Feb-2023]. 2021.

[6] F. Hu. *BipedelWalker.* `https://github.com/FranciscoHu17/BipedalWalker`. [Online; accessed 09-Feb-2023]. 2021.

[7] M. Meral. *Comparing Model-Free Deep Reinforcement Learning Algorithms on Stock Market.* `https://repository.tudelft.nl/islandora/object/uuid:852f21d1-1229-4c94-ad08-ae8a0f937d2b/datastream/OBJ/download`. [Online; accessed 09-Feb-2023]. 2021.

[8] otoro.net. *Evolving Stable Strategies.* `https://blog.otoro.net/2017/11/12/evolving-stable-strategies/`. [Online; accessed 09-Feb-2023]. 2017.

[9] R. Sutton. *Actor-Critic Methods.* `https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf`. [Online; accessed 09-Feb-2023]. 1999.

[10] H. Wei. *FORK.* `https://github.com/honghaow/FORK/blob/master/BipedalWalkerHardcore/TD3_FORK_BipedalWalkerHardcore_Colab.ipynb`. [Online; accessed 09-Feb-2023]. 2021.

[11] H. Wei and L Ying. *FORK: A Forward-Looking Actor for Model-Free Reinforcement Learning.* `https://arxiv.org/pdf/2010.01652.pdf`. [Online; accessed 09-Feb-2023]. 2021.

[12] L. Weng. *Policy Gradient Algorithms.* `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`. [Online; accessed 09-Feb-2023]. 2018.