Learning to Walk

Abstract

This paper proposes to make a bipedal agent to learn to walk by building on existing Actor-Critic methods for deep reinforcement learning. We deploy a TD3 algorithm with 2 critics and 1 actor network along with delayed policy updates by means of a replay buffer as a form of 'memory'. This recognised method was experimented with in various implementations and adapted to make the actor 'forward looking' by introducing a system and reward network to make predictions on the next state and reward of the current state respectively, given the current action. This increased both the sample efficiency and convergence allowing our agent to learn to walk within 250 episodes, and even make significant progress in more challenging environments.

1 Methodology

This paper proposes training a bipedal robot to learn to walk using the standard elements of the reinforcement learning paradigm, as an extension of a Markov Decision Process [4]. Namely, an agent in an environment attempting to behave in a manner that maximises its reward, r - in some state, $s \in \mathcal{S}$. The agent selects an action $a \in \mathcal{A}$ according to its policy (paramterised by ϕ), $\pi_\phi : \mathcal{S} \to \mathcal{A}$. This lends us to the concept of return, $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, at a given timestep,$t$. This mathematical construct as a sum of all future reward, with a discount factor, $\gamma \in [0, 1]$, indicating how far into the future our agent considers when making decisions.

If instead of some stochastic policy π_ϕ, we use some deterministic policy μ_ω, a Deterministic Policy Gradient method is obtained (cite this).

By implementing μ as the actor network, and expressing the expected return over some distribution of states and actions as $J_\pi(\phi) = \mathbb{E}_{s_i \sim p_\mu, a_i \sim \mu}[R_0]$,we can update our policy in the direction of the gradient of expected return, $\nabla_\omega J_\mu(\omega)$ to search for an optimal policy $\mu^* \in \arg\max_\mu J_\mu$, using a deterministic policy gradient algorithm [3]:

$$\nabla_\omega J(\omega) = \mathbb{E}_{s \sim p_\mu}[\nabla_a Q^\mu(s, a)|_{a=\mu(s)} \nabla_\omega \mu_\omega(s)], \tag{1}$$

where Q is the critic or value function, which is the expected return of the agent performing action a in state s, when following policy μ, i.e. $Q^\mu(s, a) = \mathbb{E}_{s_i \sim p_\mu, a_i \sim \mu}[R_t|s, a]$. While Q-learning uses the Bellman equations [1] to update the critic. This becomes unfeasible for large state spaces, and hence we rely on a deep neural network parameterised by θ as a function approximator, Q_θ. Deep Q-learning updates Q_θ using temporal difference learning with a secondary frozen target critic Q'_θ.

Deep Deterministic Policy Gradient (DDPG) is a model-free off policy algorithm

In a continuous action space, value estimates from critics in DDPG methods are over-estimated, and this is consistent in both theoretical considerations as well state of the art experiments [2]. Such overestimation is a problematic bias and can lead to the gradual accumulation of error in the temporal difference learning procedures. The Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3) avoids this using clipping techniques for the noise with $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$, along with delayed policy updates to the actor network.

In this work, we implement, study, experiment and improve TD3 through various implementations and improvements up to the FORward-looKing actor model (FORK) [5]. TD3 is a modern algorithm for deep reinforcement learning in continuous action spaces such as this one, and the main improvements from this work were achieved by combining its scope in a continuous action space with the functionality of the FORK algorithm, and tweaking and experimenting with its components.

We look at how our agent behaves with an initial implementation of TD3 that was found to be rather inconsistent before architectural improvements. These are covered in the results section.

The initial implementation of TD3 did not perform well, mainly due to a poor architecture of actor and reward networks (more neurons were necessary in each layer) and action implementation mistakes restricting the exploration of the state space. After experimenting with layers and architectural design, we implemented a more sample efficient TD3. We then looked to further ways in the literature through further experimentation to improve TD3.

The FORK algorithm introduces a system network, S_θ, and a reward netwrok, R_η, which together allow the actor from TD3 to behave in a 'forward-looking' manner. $S_\theta(s_t, a_t)$ aims to predict the next state, $\tilde{s}_{t+1}$, of the environment. It is trained using mini-batches from the agent's memory (i.e. replay buffer), while trying to minimise the smooth-l1 loss $\|\mathbb{E}[s_{t+1} - S_\theta(s_t, a_t)\|]$. Whereas $R_{\theta(s_t,a_t)}$ aims to predict the reward the agent will receive in the next timestep for the action it has just committed for the state it is currently in, so it naturally aims to minimise the mean square error loss $\mathbb{E}[\|r_t - R_\eta(s_t, a_t)\|^2]$.

	Name	**Min**	**Max**
0	Hip 1 (Torque/velocity)	-1	$+1$
1	Knee 1 (Torque/velocity)	-1	$+1$
2	Hip 2 (Torque/velocity)	-1	$+1$
3	Knee 2 (Torque/velocity)	-1	$+1$

Table 1: The continuous action space is between -1 and $+1$ so `torch.tanh` is sensible in the last actor layer. You can make tables like this using this generator.

2 Convergence Results

Our initial implementations of TD3 were successful but very slow in terms of sample efficiency and often lacked exploration. This led to the bipedal agent consistently learning to stand still and not move for the duration of the episode, favouring the subsidised of negative reward from not falling. Consequently, a good result, even after nearly 700 episodes of training was not guaranteed, with some performances being very noisy and looking very similar to a random policy for the most part of training.

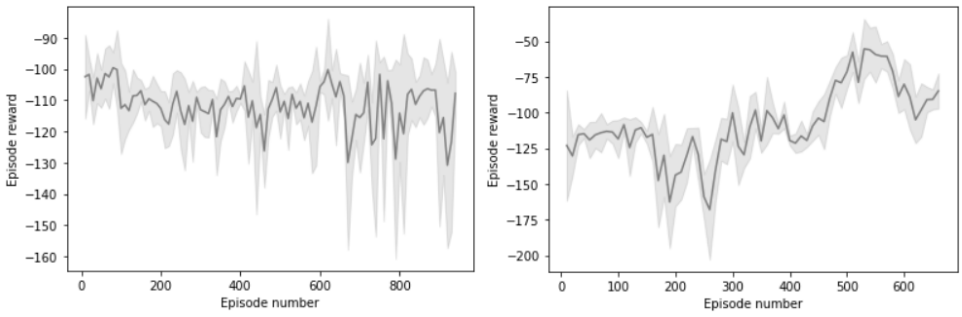

Figure 1: Initial untuned TD3 performing close to random behaviour (left) and a tuned TD3 able to converge after nearly 700 episodes.

After some progress was made on customising TD3 for a bi-pedal walker, it was noted that the agent would often come to the conclusion that standing still over the 2000 timesteps was

its best bet at not following over. This beckoned the need for more exploration and after augmenting the algorithm through other implementations (see citations in code) a better performing TD3 algorithm was obtained (Fig3 left). After modifications we found the convergence and overall smoothness of reward gain improved and the model avoided getting stuck in local minima of standing still. When running TD3 offline on a local machine for 1000 episodes, the following average score behaviour was observed (Fig 3 right).

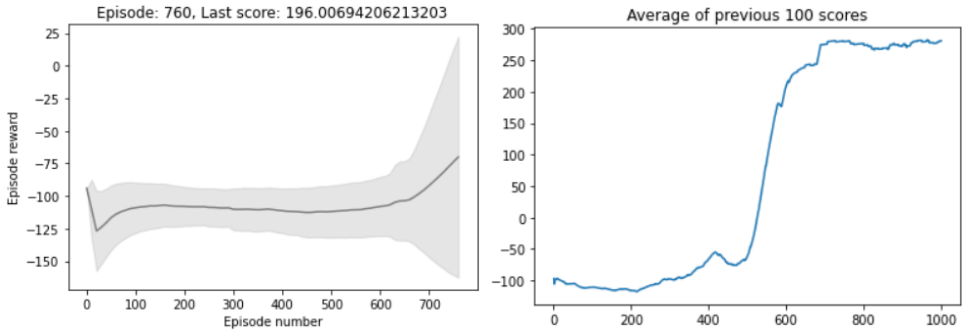

Figure 2: Better performing improved TD3 implementation (left) and the moving average of rewards from this same model during a different local machine run (right).

It was at this stage that video recordings of the agent show it learning to walk by bending its knees and taking steps forward to stabilise its centre of mass, rather than by having one leg straight and the other drag itself forward, as seen in the initial implementations of TD3. As expected from the literature, optimal performance was achieved by learning rates within the 10^{-3} order of magnitude. Changes to the size of layers in the neural networks of the actors or critics did not show improvements and in many cases worsened convergences. However, a noticeable initial dip was noted in the reward for these implementations of TD3 (Fig 2) - and this was found to be consistent on multiple runs both on virtual and local machines. This was found to be mitigated on experiments with FORK agents.

The final TD3 model is found to converge, with the agent achieving scores near 300 after the first 500 episodes.

The performance of the improved TD3 FORK, whose agent=log was submitted for this assignment is as follows:

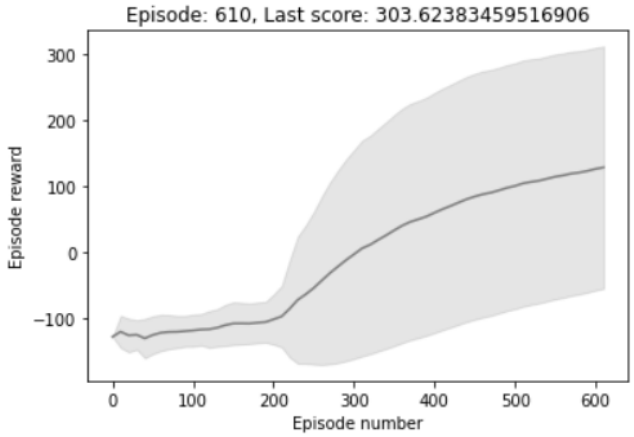

Figure 3: TD3 FORK performing over 600 episodes

Model	Reward at episode 500
Initial TD3	−49.73
Improved TD3	263.50
FORK TD3	303.35

Table 2: The performance of various models at the 500 episode mark.

3 Limitations

There is still situations where the FORK model occasionally gets stuck in the local minimum of standing still. ll though such behaviour was found much more sparsely, and more short live in the fork model.

Future Work

In the future works should aim for improvements to the novel system and reward networks. At the moment they have been implemented with intuitive objectives of foresight, but there is still potential for looking even further into future timesteps, and incorporating the concept of reward into these formulations.

References

[1] RICHARD Bellman. "Dynamic programming, princeton univ". In: *Press Princeton, New Jersey* (1957).

[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] David Silver et al. "Deterministic policy gradient algorithms". In: *International conference on machine learning*. PMLR. 2014, pp. 387–395.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Jan Uhlık. "Cooperative Multi-Agent Reinforcement Learning". In: (2021).