Learning to Walk With TD3

Anonymous author

Abstract

This report proposes a method to learn to walk by probabilistically sampling actions via a TD3 policy, utilising an actor and a pair of critics. This paper explores the methodology in further detail, reviews some results, and suggests routes for improvement, for instance a Prioritised Experience Replay buffer.

1 Methodology

The corresponding implementation explores training a bipedal robot to learn to walk by sampling actions a with a TD3 policy (Twin-Delayed Deep Deterministic Policy Gradient), as originally proposed by Dankwa and Zheng [1]. The agent acts on the Bipedal Walker's action space, as shown in Table 1.

	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 robot's continuous action space, between -1 and $+1$.

In brief, the TD3 policy operates in the following way:

- Instantiate a series of neural networks - an actor model π_ϕ & target pair $\phi\prime \leftarrow \phi$, and two critic models $Q_{\theta_1}, Q_{\theta_2}$& corresponding target pairs $\theta_1\prime \leftarrow \theta_1, \theta_2\prime \leftarrow \theta_2$,.
- Also create a structured memory buffer $\mathcal{B}$ to store prior transition experience for training use.

  For T timesteps do:

- Select an action from the action space using the agents network as an exploration policy $a \sim \pi_\sigma(s)$ along with the introduction of some random Normally distributed noise. Then make a step with this action in the environment and store the transition in the buffer. Update episode reward with the action reward.

  Sample batch from $\mathcal{B}$ and do the following:

- Compute each critic loss (via mean squared error of the critic network versus target critic network) and backpropogate, then optimise parameters with 'Adam' SGD. $\theta_i \leftarrow \mathrm{MSE}(Q_{\theta_1}(s, a\prime), \theta_1\prime) + \mathrm{MSE}(Q_{\theta_2}(s, a\prime), \theta_2\prime)$
- Every other iteration, update Actor model where loss is the negation of the gradient descent through the critic model. $update = \frac{1}{batchsize} \Sigma \nabla_\phi(s, a)$
- If buffer is full,train the agent based on the rewards learnt, randomly sampling from the replay buffer.

After experimentation, I introduced a variable number of timesteps per episode to the implementation, such that there are a series of initial exploration phases (1000 up to episode 50, then 1500 up to episode 100, and then settling at 2000). The goal of this strategy was to quickly build up the replay buffer with experiences for future use, and it proved successful with faster convergence observed.

The TD3 architecture used is summarised in Figure 1.

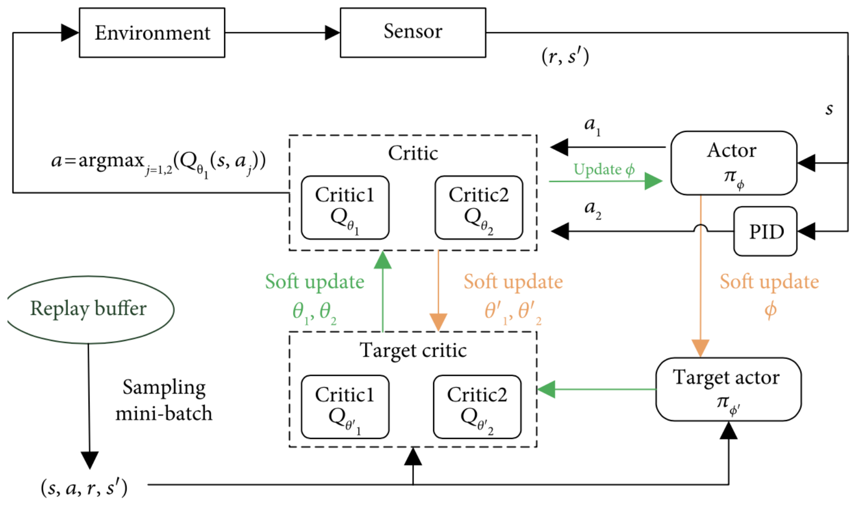

Figure 1: The TD3 architecture [3]

2 Convergence Results

After time, the model converges with achieved results consistently near 300, as shown in Figure 2. This is standard behaviour exhibited across repeated training cycles, giving confidence that the model will always eventually converge.

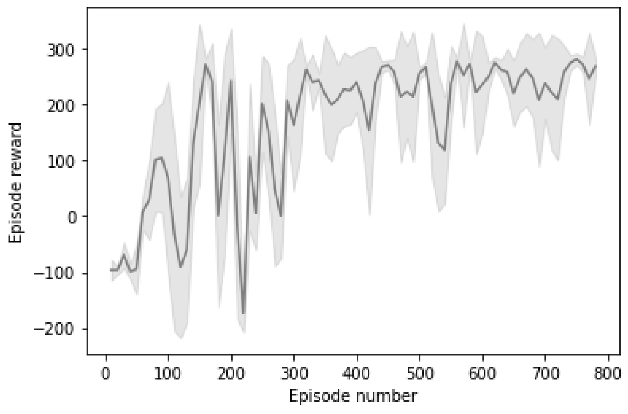

Figure 2: Performance of model over training episodes

Notably, the agent appears to plateau just below a score of 300 (aside from the odd outlier) - this can be seen to hold up to episode 2000 in the agent log. This is likely a systemic limitation, potentially due to the network structures of the agent or critics, and would need these to be reworked to achieve better results.

3 Limitations

While the model here does converge, it is not consistently stable - it can be seen there are a series of points where performance drops off rapidly, for instance at roughly 220 episodes. This is likely due in part to the way the replay buffer is being handled, with valuable experiences being discarded and the agent loosing memory of suitable sample action pairs. The basic replay buffer being used selects experiences entirely stochastically, without any regard for how useful said experience is to learn from, and is hence limiting training (convergence) time, and potentially also final performance accuracy. There is also scope for faster convergence -

Future Work

To improve this system further, a Priority Experience Replay buffer should be introduced. A PER seeks to sample from experiences in such a way that more valuable experiences (e.g. higher reward) are more likely to be sampled, whilst still guaranteeing that every experience in the buffer has a non-zero chance of being sampled. Schaul et al [2] show that a PER can yield major perforamnce and stability benefits, with unchanging error frequency over training iterations.

A PER was attempted in this work (included for reference at bottom of code), however was unfortunately unsuccessful due to difficulties in integration with the buffer sampling techniques used. To achieve this, a major refactoring of the network should be undertaken.

It would also be advisable to experiment more with network layer sizes in a way that seeks to increase the final achieved result.

References

[1] Stephen Dankwa and Wenfeng Zheng. "Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent". In: Aug. 2019, pp. 1–5. DOI: `10.1145/3387168.3387199`.

[2] Tom Schaul et al. *Prioritized Experience Replay.* 2015. DOI: `10.48550/ARXIV.1511.05952`. URL: `https://arxiv.org/abs/1511.05952`.

[3] ZhiBin Zhang et al. "Model-Free Attitude Control of Spacecraft Based on PID-Guide TD3 Algorithm". In: *International Journal of Aerospace Engineering* 2020 (Dec. 2020). Ed. by Marco Pizzarelli, pp. 1–13. DOI: `10.1155/2020/8874619`. URL: `https://doi.org/10.1155/2020/8874619`.