Application of TD3 algorithm to the BipedalWalker-v3 environment

Luka Utješinović
lu0801@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Slovenia

Luka Boljević
lb7093@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Slovenia

ABSTRACT

First we provide a basic overview of actor-critic RL methods. We then present the TD3 algorithm as an extension of the DDPG algorithm. We explore the BipedalWalker-v3 environment and we successfully teach an agent how to walk. We discuss the obtained results in the conclusion.

KEYWORDS

Reinforcement learning, deep reinforcement learning, actor-critic methods, deterministic policy gradient, robotics

1 PRELIMINARIES

Primary modeling paradigm in RL is modeling with Markov decision processes. Formally, a **Markov decision process** is a tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the **state space**
- $\mathcal{A}$ is the **action space**
- P is the **transition probability distribution**, $P^a_{s's} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- $\mathcal{R}$ is the **reward function**, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ where R_{t+1} is a random variable which represents the reward returned by the environment at time step $t+1$.
- $\gamma \in [0, 1]$ is the **discount factor**

With $G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}$ we denote **discounted return** from time step t. **Policy** π is a probability distribution of actions over states, that is $\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$. If for every state s there exists an action a such that $\pi(a \mid s) = 1$, we say that π is a **deterministic policy**. Otherwise, π is a **stochastic policy**. It is easy to observe that G_t depends on the action chosen at each time step, and therefore depends on the choice of policy π. One important quantity for a policy π is the **action value function**, which is defined as $Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$.

2 ACTOR-CRITIC MODELS

Let us briefly cover actor-critic methods. These methods combine value (action value) function approximation methods and policy gradient methods. **Critic** is concerned with approximation of action value function, and the underlying model uses a set of parameters w to approximate the action value function. The objective function of the critic is typically chosen to be mean squared error.

On the other hand, **actor** updates a parametrized policy π_θ using the direction of **policy gradient** while at the same time consulting the critic for the action value function. For particular domains (like robotics), it turns out that it is easier to train a deterministic policy rather than a stochastic one, while with the standard policy gradient framework the trained policy is stochastic. Authors of [3] show that for a deterministic policy π_θ and an objective function $J(\pi_\theta) = \mathbb{E}[G_1 \mid \pi_\theta]$, under certain regularity conditions the gradient can be computed with the following expression:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim \rho^{\pi_\theta}} \left[\nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s, a) \Big|_{a = \pi_\theta(s)} \right] \tag{1}$$

where ρ^{π_θ} is the stationary distribution of the underlying Markov decision process given a deterministic policy π_θ.

3 THE TD3 ALGORITHM

One interesting algorithm from the actor-critic family is **Deep Deterministic Policy Gradient - DDPG** [2], which uses fully-connected neural networks for actor and critic. Two additional networks are kept, and these networks are called **target networks**. Initially they represent exact copies of actor and critic networks, however their weights slowly follow the weights of original actor and critic networks during the execution of the algorithm. Authors reason that this greatly improves the stability of learning. The **replay (experience) buffer** $\mathcal{B}$ is also introduced, a data structure which stores a finite number of transition tuples (s, a, r, s'). All weight updates are done after sampling a batch of transition tuples without repetition from the replay buffer. This is done because sequences of these tuples are highly correlated, which makes gradient-based optimization algorithms perform poorly. $\mathcal{B}$ was implemented as a simple queue.

For this environment however, DDPG showed poor performance, after which we implemented **Twin Delayed DDPG - TD3** [1] which represents a modified version of DDPG. In essence, the authors reason that introducing an additional critic network and updating actor and target networks after a period of $d \geq 2$ time steps tends to reduce the overestimation bias of DDPG. The pseudocode for TD3 is given in algorithm 1.

4 BIPEDALWALKER-V3 ENVIRONMENT

To test our implementation, we used the BipedalWalker-v3 environment from the popular RL Python library, Gym. This environment simulates a robot walking on flat terrain. **Reward** is given for moving forward, with a max of 300 points If the robot falls, it gets a reward of -100. Applying motor torque costs a small amount of points, which encourages agents to move as efficiently as possible. A more detailed description of the environment can be found on this link. The architecture of actor and critic networks can be seen on figure 1, while the used hyperparameters can be found in the following file.

Algorithm 1 TD3

```
1:  Initialize critic networks $Q_{w_1}, Q_{w_2}$ and actor network $\pi_\theta$ with random weights $w_1, w_2, \theta$
2:  Initialize weights target networks $w'_1 \leftarrow w_1, w'_2 \leftarrow w_2, \theta' \leftarrow \theta$
3:  Initialize the replay buffer $\mathcal{B}$
4:  for $t = 1, T$ do
5:      Select an action with exploration noise $a \leftarrow clip(\pi_\theta(s) + \epsilon, -c, c), \epsilon \sim \mathcal{N}(0, \sigma)$
6:      Execute action $a$ and observe reward $r$ and new state $s'$
7:      Store tuple $(s, a, r, s')$ in the replay buffer $\mathcal{B}$
8:      Sample a mini-batch of N transitions $(s, a, r, s')$ from the replay buffer
9:      $\hat{a} \leftarrow \pi_{\theta'}(s') + \epsilon, \epsilon \sim clip(\mathcal{N}(0, \hat{\sigma}), -\hat{c}, \hat{c})$
10:     $y \leftarrow r + \gamma \min_{i=1,2} Q_{w'_i}(s', \hat{a})$
11:     Update critics following the mean squared error loss on the current mini-batch $w_i \leftarrow \arg\min_{\theta_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2$
12:     if t mod d then
13:         Update parameters $\theta$ following averaged deterministic policy gradient from equation 1:
                $\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum \nabla_a Q_{\theta_1}\big|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
14:         Update target networks:
                $w'_i \leftarrow \tau w_i + (1 - \tau) w'_i, i = 1, 2, \theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
15:     end if
16: end for
```

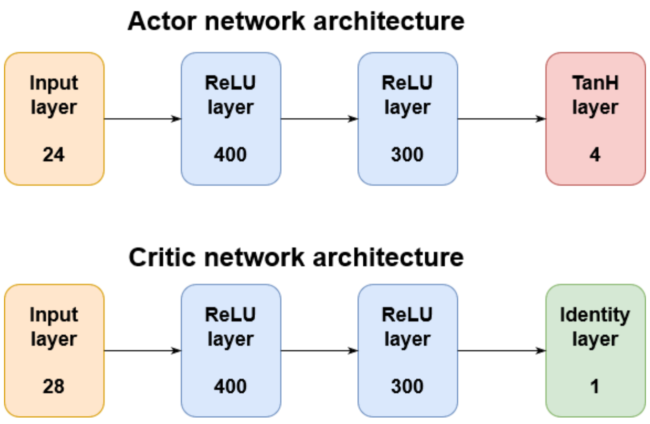

Figure 1: Neural network architectures. Tanh was the activation of choice for the output layer of the actor network because action space is $[-1, 1]^4$

We repeated the training process 15 times, and kept track of obtained reward for each episode. The obtained results are seen in figure 2.

Confidence regions were constructed using the **asymptotic normality argument** ($\mu \pm 1.96SE$) which is very crude, but still it helps us to get a sense of inherent variability of TD3.

In the first 100 episodes, confidence regions are pretty narrow, which is expected because we expect the agent to perform poorly in the beginning. As it starts to get better however, confidence regions start getting wider, which is contributed to the inherent variability of TD3.

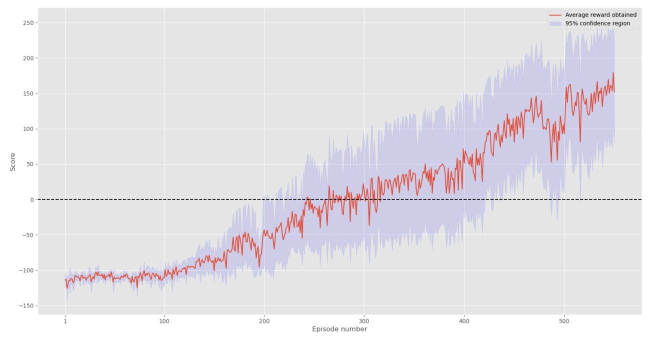

Figure 2: The results of our testing. The red line represents the average reward of the 15 agents, while the blue region is a 95% confidence interval.

We also notice a steady climb of average rewards, but we also spot two dips around the 200 and 500 episode mark. This could be due to an interesting phenomenon called **catastrophic forgetting** - neural networks to forget all previously learned information upon learning new information. This could perhaps be suppressed by increasing the size of the replay buffer (which is set to 10^6). We did not experiment with this because hardware resource requirements were already quite demanding. Overall however, we would say that TD3 gave good results for this environment.

5 CONCLUSION

TD3 is a very powerful algorithm for continuous action space environments in which the policy of interest should be deterministic. The only downside of this algorithm observable from this experiment would be it's variability. It could perhaps be suppressed by repeating the training process multiple times and averaging the obtained agents at the end, which could be researched in the future. Further work may also include testing TD3 for other continuous space/continuous action environments, like Lunar Lander environment.

REFERENCES

[1] Fujimoto, Scott and van Hoof, Herke and Meger, David. 2018. Addressing Function Approximation Error in Actor-Critic Methods. https://doi.org/10.48550/ARXIV.1802.09477
[2] Lillicrap, Timothy and Hunt, Jonathan and Pritzel, Alexander and Heess, Nicolas and Erez, Tom and Tassa, Yuval and Silver, David and Wierstra, Daan. 2015. Continuous control with deep reinforcement learning. *CoRR* 1 (09 2015).
[3] Silver, David and Lever, Guy and Heess, Nicolas and Degris, Thomas and Wierstra, Daan and Riedmiller, Martin. 2014. Deterministic Policy Gradient Algorithms. *31st International Conference on Machine Learning, ICML 2014* 1 (06 2014).