Exploring Actor-Critic Algorithms for Controlling a Biped Robot in Simulation

Pedro Batista
pedro.peres.batista@ubi.pt

André Correia
andrerspcorreia@gmail.com

Luís A. Alexandre
luis.alexandre@ubi.pt

Departamento de Informática
Universidade da Beira Interior
NOVA LINCS
6201-001, Covilhã, Portugal

Abstract

Reinforcement Learning (RL) is a field of machine learning which studies how agents can learn the best actions to take in a certain state of the environment. Differently from other fields of machine learning where agents learn from input-output pairs, in RL the agent needs to find the best actions to reach the desired outputs in a trial-and-error cycle. Here we study the behavior of several Actor-Critic [5] RL algorithms on the problem of learning to control a biped robot. Solving this problem is important to help develop robots that are able to perform some human tasks, such as some that involve dangerous situations. From the experiments made we concluded that the Soft Actor-Critic (SAC) with dynamic alpha coefficient presented the best results, although it is also the most computationally demanding algorithm among the tested ones.

1 Introduction

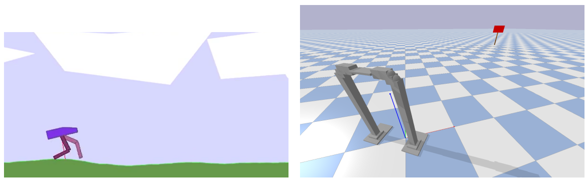
The main goal of teaching robots how to walk bipedally is so they can perform the tasks we humans execute, including the most dangerous ones, where human lives are at risk. This objective of teaching a robot to walk bipedally can't be accomplished by a traditional computer algorithm without machine learning methods because of the complexity and number of different scenarios the agent could face, adding the infinite number of actions it could take. In practice, it would be like writing down a huge set of rules which tells the robot what actions to take (for example, which torque forces to apply to each joint motor) for each different situation it could encounter.

We studied three different Actor-Critic algorithms: Deep Deterministic Policy Gradient (DDPG) [6], Twin Delayed DDPG (TD3) [2] and SAC [3]. They rely on the Bellman equation, where the agent has, at least, a policy, named actor μ_θ and a critic neural network Q_ϕ, with parameters θ and ϕ, accordingly. The policy tries to predict the best action for a given observation. The critic evaluates how good an action is for a given state. The learning process also requires a memory buffer where the agent stores previous transitions containing: observation, action taken, reward, next observation and whether the next state is terminal or not. To learn both the policy and critic parameters, the agent samples a batch of transitions from the memory buffer. The policy is trained based on the critic's feedback, causing it to take actions that are expected to get higher rewards, thereby successfully performing the task. The critic is updated based on the difference between its predictions and the actual rewards. To prevent the online critic and policy networks from overfitting, we employ the exponential moving average technique, which uses target networks to delay the update of the online networks.

2 Actor-Critic Algorithms

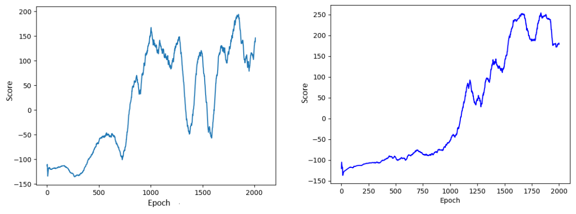
The DDPG algorithm makes use of two online and two target policy and a critic neural networks. The TD3 algorithm is an improved version of DDPG. It solves the overestimation of the critic values and the overfitting of the policy. To do so, it brings three main improvements: 1) Policy smoothing: The noise is not only applied to the policy actions but also to the target policy during training to avoid overfitting; 2) Two or more critic networks: This method reduces the likelihood of overfitting, choosing the minimum value predicted by the critic networks to calculate the gradient; 3) Delay of policy update: The policy update delay gives the critic networks time to converge.

Like the TD3, SAC uses, at least, two critic networks. The biggest difference is that SAC introduces entropy to policy learning, avoiding the need for a target network. The entropy in SAC promotes exploration, encouraging randomness in the actions to better explore the action space. In the SAC algorithm, the agent receives an additional small reward equivalent to the policy entropy in each time step, changing the reinforcement learning problem:



(a) Bidimensional environment. (b) Three-dimensional environment.

Figure 1: Environments used for the biped robot experiments.



(a) DDPG experiment results. (b) TD3 experiment results.

Figure 2: DDPG and TD3 experiment results.

$$\pi^* = \arg\max_{\pi} \left(\sum_{t=0}^{\infty} \gamma^t (r + \alpha H(\pi(s_t))) \right) \tag{1}$$

where π is the stochastic policy, γ is the discount factor, r is the reward, α is the hyperparameter which defines the trade-off between exploration and exploitation, H is the entropy, and s_t is the observation at time t.

3 Experiments

3.1 Bidimensional Simulation Experiments

The 2D experiments took place in OpenAI Gymnasium [1] (see Fig. 1(a)). The robot has a base, two legs, four joints and a LIDAR sensor, which produces an observation with the following information: base angular velocity, horizontal velocity, vertical velocity, joints position, angular joint velocities, leg contact with the ground information and LIDAR readings. The action representation is a simple $\mathbb{R}^4$ tuple which contains torque forces for the four joints, each inside the range $[-1,1]$.

We ran the Actor-Critic algorithms for 2000 epochs, memory batches of size 64, fully connected neural networks with two layers, each with 512 neurons, a discount factor of $\gamma = 0.99$, learning rate of 0.001 for the actor and 0.002 for the critic, Polyak constant $p = 0.995$ and mean-zero Gaussian noise with a standard deviation $\sigma = 0.1$.

DDPG, although being able to get good rewards, does not present a smooth learning curve (Fig. 2(a)). This problem occurs due to the overestimation of the critic's values because of the greedy nature of the critic function. Not only is the critic network greedy, but DDPG doesn't apply policy smoothing, causing the policy to get stuck on local minima.

The TD3 algorithm proved to be more stable than DDPG. The stable results were possible due to the three main changes made to the DDPG algorithm: target policy smoothing, two critic neural networks and policy update delays. Fig. 2(b) contains the TD3 practical results.

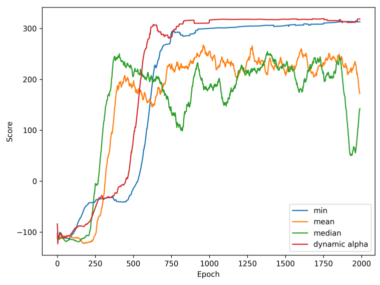


Figure 3: SAC results on the 2D environment.

Table 1: Algorithms performance comparison

Algorithm	Final score	Execution time (h)
DDPG	175.7	05h27
TD3	209.2	11h07
SAC (min)	313.3	19h29
SAC (median)	142.3	11h32
SAC (mean)	172.6	11h40
SAC (dynamic)	318.9	26h02

With the introduction of entropy, the SAC algorithm brought excellent results, outperforming DDPG and TD3 in a quarter of the epochs. The experiment setup for SAC includes some new parameters: entropy coefficient $\alpha = 0.01$ and $\sigma \in [-20, 2]$. The results can be seen in Fig. 3 (min line).

The Actor-Critic algorithms usually use, at least, two critic neural networks to avoid overestimation of the critic value. These algorithms often select the minimum value predicted by the critics, to avoid overestimating the true critic value. We tested other evaluation mechanisms, such as collecting the mean and the median between the predicted critic values. We can conclude that the use of the minimum value leads to a more stable learning with higher rewards in the long term. This happens because the use of the minimum value prevents the algorithm from converging to bad local minima. On the contrary, the use of the mean and median values promotes high rewards (above 100) in the short run due to premature exploitation of non-optimal actions, but the performance quickly falls because the networks end up converging to a bad local minimum.

The SAC algorithm can manage the trade-off between exploration and exploitation with a good performance due to the introduction of entropy. The higher the entropy, the higher the exploration. The downfall of this approach is the need to adjust the α coefficient manually, requiring many experiments to understand what could be the best value for this coefficient in each scenario.

To automatically adjust the α coefficient, we make use of a neural network to predict the coefficient value [4]. To update this neural network's parameters, we have to calculate the gradient using the cost function (2), where s_t is the observation at t sampled from the memory buffer and $\mathcal{H}$ is a constant which is equal to the action space dimension times -1.

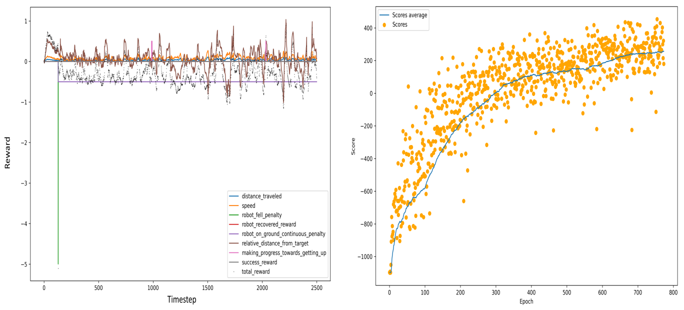
$$J(\alpha) = -\alpha(\log \pi(s_t)) - a\mathcal{H} \tag{2}$$

Fig. 3 showcases the performance difference between each SAC algorithm tested in the context of this work.

From Table 1 we can conclude that the best-performing algorithm is the SAC with the dynamically adjusted entropy coefficient. Thanks to the dynamic trade-off between exploration and exploitation, the agent can learn and adapt to the environment more efficiently, although this comes with an increased computational cost when compared to the other SAC versions, since it uses another neural network for estimating α.

3.2 Three-dimensional Simulation Experiments

We built a 3D robot with 10 joints in the *Autodesk Fusion 360* software and converted it to a *URDF* file, allowing us to test the robot in the *PyBullet* physics simulator. In this environment, we were challenged to focus on the observation and action representations, as well as the reward function architecture. The three-dimensional environment can be seen in the Fig. 1(b), where the robot and a flag representing the target destination, are shown. The agent observation space representation is the following: position, orientation, rotation and velocity of the joints, the robot linear



(a) Reward function components. (b) SAC results.

Figure 4: Experiments in the 3D environment.

and angular speeds, the relative distance to the target destination and foot contact with the ground information. The action space representation consists of a $\mathbb{R}^{10}$ tuple containing each joint rotation angle. Because each joint has different rotation angle limits, a linear transformation is applied to convert the actions to the range $[-1, 1]$ returned by the policy, to each joint limit. This is achieved with the function $T : \mathbb{R}^3 \to \mathbb{R}$ in equation (3).

$$T(x, a, b) = (b - a)(x + 1)/2 + a \tag{3}$$

The reward function is designed to help the agent evaluate its actions during training. For this purpose, it takes into account the following parameters: traveled distance in the last time step, robot speed, penalty for falling, a penalty for being in the fallen state, reward for trying to get up, reward for getting up, reward for having its feet in contact with the ground, relative approximation to the target destination and reward for reaching the target destination. The different reward components can be visualized in Fig. 4(a), where we can see the rewards in the course of a 2500 time steps epoch.

To run the three-dimensional experiments, we used the SAC algorithm with the dynamically adjusted entropy coefficient. The experiment results can be seen in Fig. 4(b). They show that this algorithm is able to learn how to control the robot in this more complex environment, also thanks to the careful definition of both the observation and action spaces.

4 Conclusions

We implemented three Actor-Critic algorithms according to their publication descriptions, and used them to solve the biped robot problem. We also implemented other variants, including one that obtained the best performance in the 2D experiments. We concluded that not only the learning algorithm is important, but also the inputs it receives from sensors and the environment. If the observation space doesn't represent the real environment state well enough, or the reward function from the environment isn't returning useful feedback, then our agent won't be able to learn regardless of the learning algorithm used. We also saw that the best results were obtained using SAC with dynamic α adjustment, although at a higher cost when compared to the remaining SAC versions.

References

[1] Openai gymnasium. `https://gymnasium.farama.org`.

[2] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proc. 35th ICML*, 2018.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. 35th ICML*, 2018.

[4] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Robotics Science and Systems*, 2019.

[5] Cheng shuo Ying, Andy H. F. Chow, and Kwai-Sang Chin. An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand. *Transportation Research Part B: Methodological*, 2020.

[6] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proc. 31st ICML*, 2014.