2023-2024 FALL SEMESTER

CS 449

HOMEWORK 3

Group 10

Members:

Arda YILDIZ - 22003093

Asım Bilal AK - 21802887

Erkin AYDIN – 22002956

Mustafa Kaan KOÇ - 21903358

1. A Brief Explanation of the TD3 Algorithm

The TD3 (Twin Delayed DDPG) algorithm extends the DDPG framework, targeting the issue of overestimation bias in value-based reinforcement learning. The overestimation bias can result in suboptimal policies, as the agent may overvalue certain actions or states. Hence, the goal is to overcome the overestimation bias to achieve more accurate results. To mitigate the impact of overestimation bias, multiple critics can be used. Thus, the TD3 algorithm employs the usage of a pair of critics instead of a single one, to address the effects of overestimation bias by taking the minimum value between the two critics' estimates. Moreover, the TD3 algorithm introduces delayed policy updates to reduce per-update error, allowing the value function to converge more effectively before updating the policy. Finally, these two features of the TD3 algorithm combined results in stability, more performance and higher accuracy.

2. Summary of the Implementation

The implementation in this homework is a representation of the reinforcement learning with Twin Delayed Deep Deterministic policy gradient (TD3) algorithm. A Lunar Landing environment is simulated from the Gymnasium library.

Hyperparameter Configuration

Hyperparameter class contains hyperparameters that describe the specific features of the environment and methodology of the learning. It identifies some aspects such as render, gamma, max_train_steps. While the render parameter corresponds if the training will be done with the visual rendering, the gamma parameter sets if the training will focus on future steps or the immediate step. On the other hand, max_train_steps declares the maximum training number.

Main Function

In the main function, we have specified the Gym environment for both training and evaluation. On the other hand, we have built 2 different environments as ‘env’ and ‘eval_env’. Then we have done the hyperparameter initialization and applied seeding.

Hyperparameter Initialization: Maximum action value, dimensions of the action space and state are taken from the environment and initialized as the hyperparameters.

Seeding: We have set a seeding value at the beginning. Next, we have incremented it by 100 in each episode of the training as it is done in the paper of the TD3.

TD3 Agent Initialization:

While setting up a TD3 agent, hyperparameters are tuned. The agent comprises two components: An "actor" network (to determine the agent's actions based on the current state), and a pair of "critic" networks (to estimate the rewards for given state-action pairs). Therefore, learning rates of these two hyperparameters are initialized/tuned in the hyperparameters section. Moreover, the agent includes a "replay buffer" to store and randomly sample past experiences. Hence these initialization steps support the TD3 agent with the necessary structures for learning and making decisions in an environment. In addition, there is a select_action method for the agent which chooses a deterministic action or an action with added noise, depending on the specification of the mode. Moreover, the train method updates both the actor and critic networks, leading to the training of the agent.

Training and Evaluation Loop:

The agent is trained in different environments. To add more randomness to the environments, the random seeds are fed to the system. Hence, the purpose of training the agent in different environments to achieve more accurate results is satisfied. Moreover, we train the agent 50 times every 50 steps rather than 1 training per step to achieve better results. One final remark, we are setting the render mode to False, since performance drops when we render the training process and we spend more time when the performance drops.

Utility Functions

The implemented model consists of a neural network comprising three linear layers. It uses the hyperbolic tangent (tanh) activation function, facilitating the mapping of states to corresponding actions.

The Double Q Critic architecture includes two distinct critic networks. Each network includes three linear layers with Rectified Linear Unit (ReLU) activations, shown as max(0,x). These networks estimate the Q-value related to the given state-action pair.

Evaluate_policy runs the policy for a specified number of episodes. After that, it returns the average reward according to TA’s recommendation. 1 episode includes 1000 steps with respect to recommendation.

The 'Reward_adapter' function serves the purpose of modifying the reward. This function adjusts the reward to approach the objectives of the evaluation process.

Execution Flow

1. Initialization: The initialization process includes the tuning of the hyperparameters for both the environmental settings and the TD3 agent.
2. Training Loop: The training loop contains interactions between the agent and its environment. During these interactions, the agent accumulates knowledge and periodically repositions itself due to the training knowledge it acquires. In this process, a rendering option is available, in case the user wants to visually observe the training process. However, the rendering mode is not used since it reduces the performance, as mentioned before.
3. Evaluation: In this process, the agent's performance is assessed with tests conducted at predetermined intervals to gauge its proficiency and adaptability.

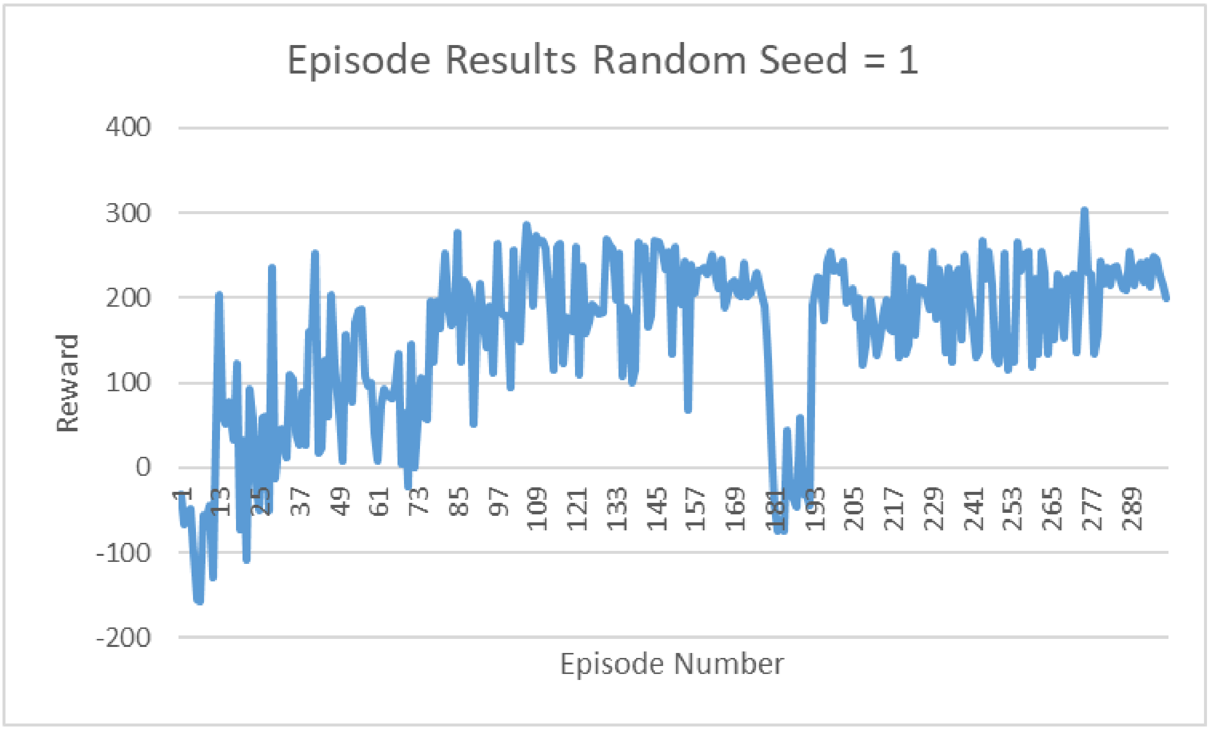

Figure 1: Reward vs episode number when seed is 1

- Last 100 episode average is: 200,3663

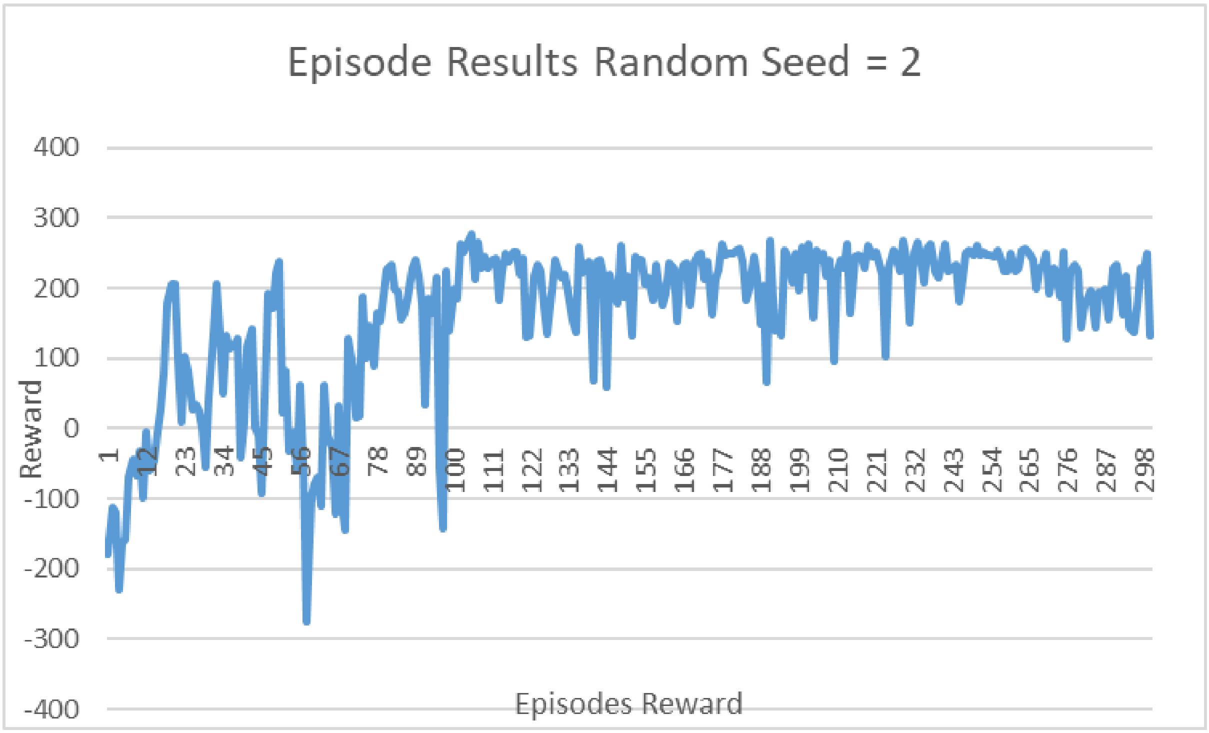

Figure 2: Reward vs episode number when seed is 2

- Last 100 episode average is: 220,8218

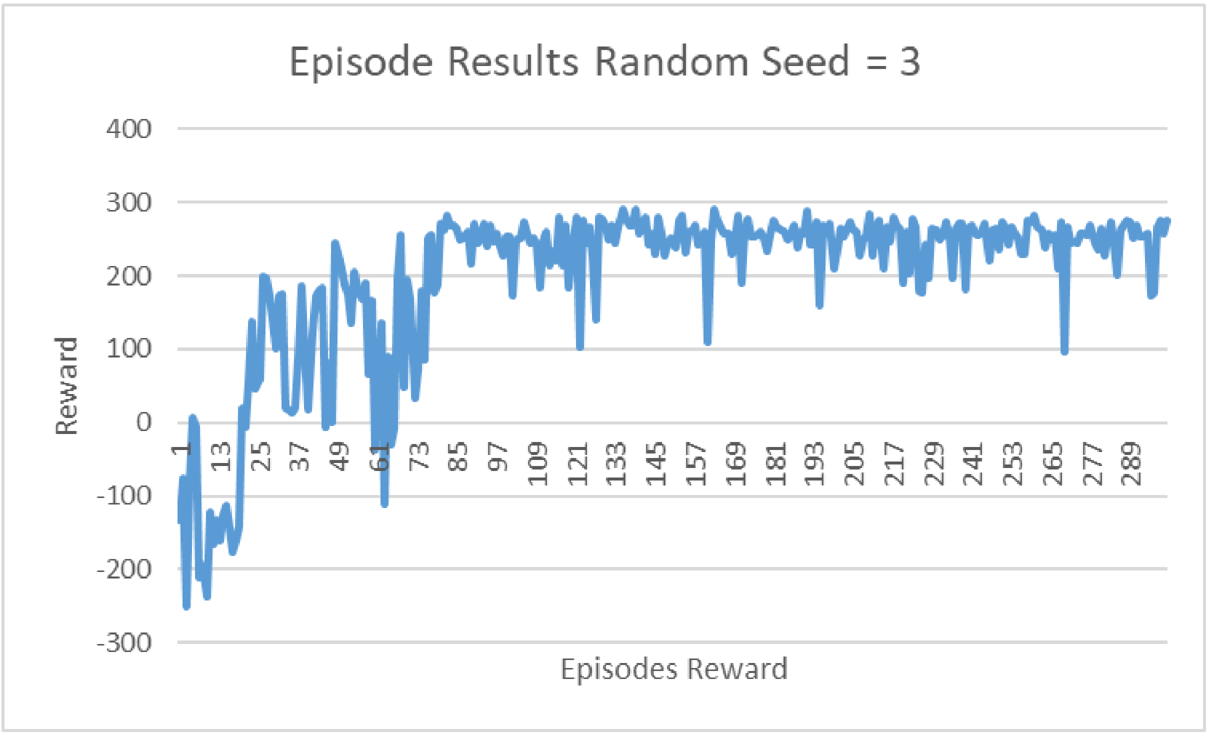

Figure 3: Reward vs episode number when seed is 3

- Last 100 episode average is: 249,297

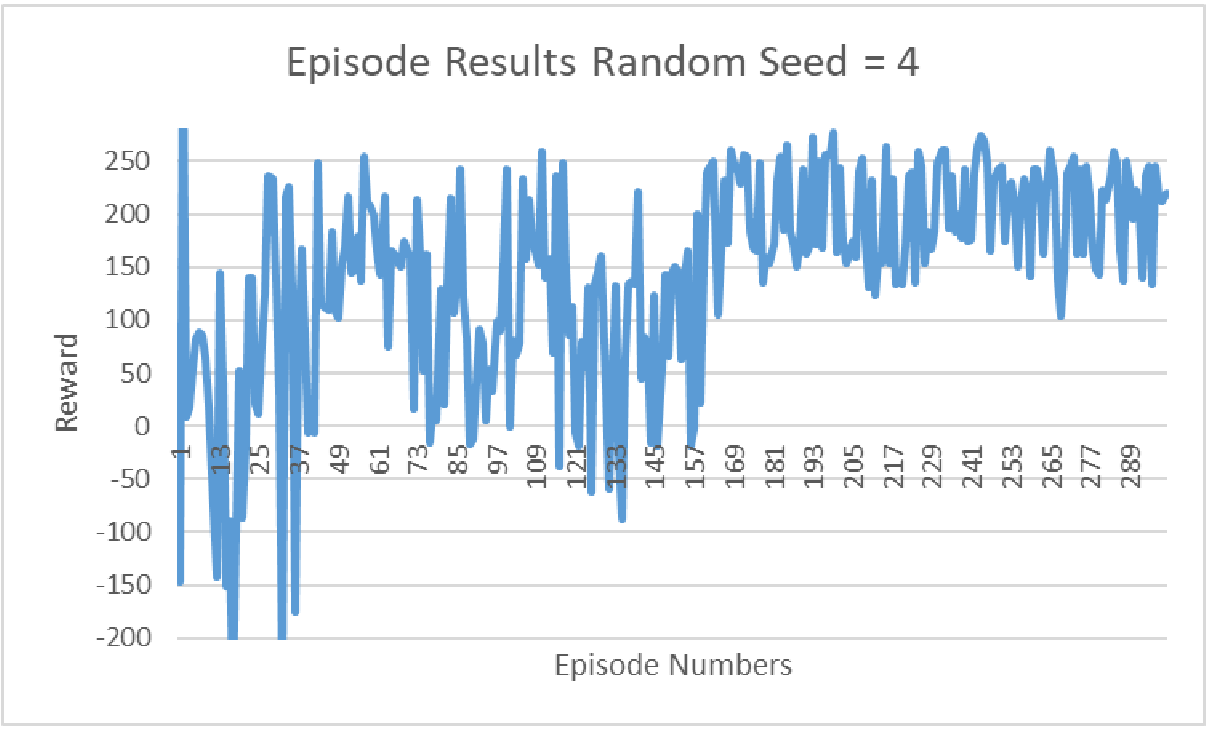

Figure 4: Reward vs episode number when seed is 4

- Last 100 episode average is : 202,7228

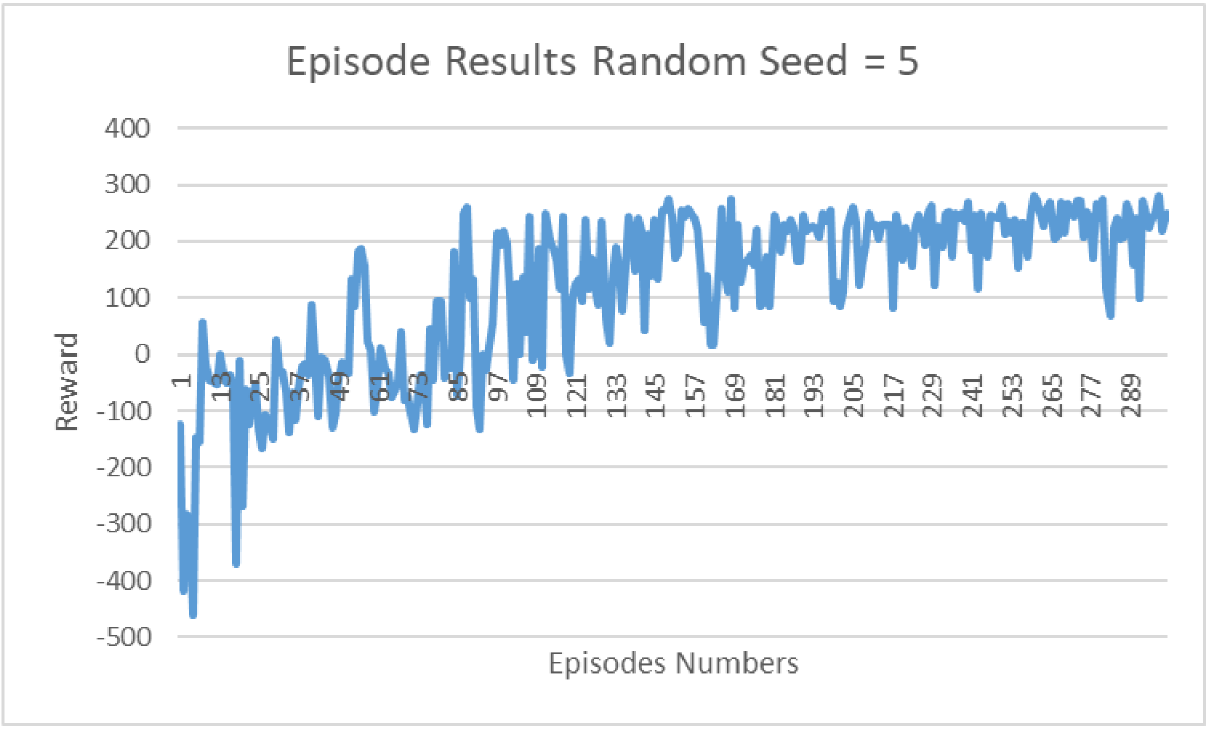

Figure 5: Reward vs episode number when seed is 5

- Last 100 episode average is : 218,6931

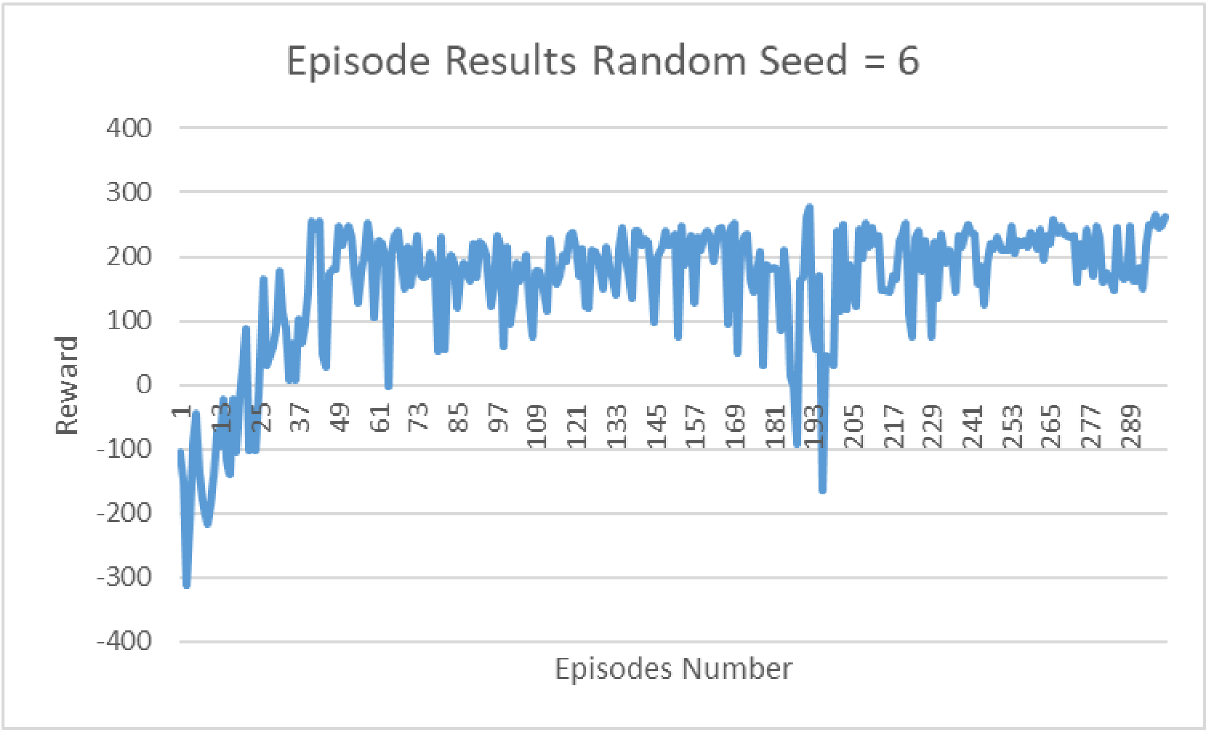

Figure 6: Reward vs episode number when seed is 6

- Last 100 episode average is : 203,39

Average of all 6 runs: 218,302

3. Analysis of the results

With respect to the graphs, although the reward value fluctuates, its average reward value increases as the episode number rises for each seed number we have trained for. Specifically, with respect to the Figure1, Figure 2, and Figure 3 there appears an increase in the average reward values with respect to increasing seed values. The increment in reward average occurs until some specific episode values. For instance in Figure 1, it is observed that episode numbers around 170 have a massive drop for reward values, going from almost 250 to nearly -100. This radical change in the reward value seems to be caused by the randomness of the changing seed values. After that point, it starts to converge to a value between 200 and 250, and randomness of the seed value seems to lose its impact on the reward values.

At the beginning of each run, the reward function was not stable and could take various values. However, when we kept training the agent, the reward values became more stable. Moreover, getting a high reward means our actor will act in a more reliable way. So, in general, the reliability of the actor rises as we increase the episode number.

4. Discussion

As a result of our experiments, it was observed that there is a direct proportion between the number of experiments and the results obtained. While our seeds are 1 and 4, there seems to be a lot of fluctuation and our average episode-reward average seems to be lower. However, we achieved relative stability after approximately 200K trials. As it can be seen, we had higher average results in our training for seed numbers 2, 3 and 5. Moreover, Figures 2, 3, and 5 (corresponding to seeds 2, 3, and 5) display more stable results for reward values, as the fluctuations are smaller compared to Figure 1, 4, and 6. As it can be understood from here, if the number of training increases, stability increases and the desired values are reached which is a reward of 200. Based on this, changing the number of seeds changes the patterns for reward values. However, the results displayed in the graphs highlight that the number of trainings is the cornerstone for achieving higher reward values. Last but not least, although there is no particular pattern for having a stable graph for increasing seed values, there appears to be a randomness caused by changing seed values which seems to be the cause of the changing stability in between different graphs above.

5. References

https://github.com/sfujim/TD3. Accessed on 19, November 2023.
https://arxiv.org/abs/1802.09477. Accessed on 19, November 2023.
https://github.com/XinJingHao/TD3-Pytorch. Accessed on 19, November 2023.