Reinforcement Learning - Assignment 3

Naman Tejaswi
University at Buffalo
namantej@buffalo.edu

Shreyans Pathak
University at Buffalo
spathak3@buffalo.edu

Abstract

In this assignment, our aim is to implement the actor critic approximation algorithms discussed in the class onto the OpenAI Gym based environments. The Assignment is divided into two parts, in the first part we have to select an environment from the suggested list of environments and run a base actor critic algorithm and then in the second part, we have to apply the same algorithm on slightly more complex environments. We also need to extend the base algorithm to a more advanced version of it and discuss the overall results and challenges faced.

"We certify that the code and data in this assignment were generated independently, using only the tools and resources defined in the course and that we did not receive any external help, coaching or contributions during the production of this work."

1 Algorithm Details

In this section, we will briefly go through the algorithm we have implemented for the project, along with all the necessary details required. First, let us quickly go through the actor critic algorithm in the next section.

1.0.1 Actor Critic Methods

Actor Critic methods in Reinforcement learning utilize the combination of two different variety of algorithms - value optimization and policy optimization. The hybrid model on which actor critic algorithms work combine the well known value based algorithms like Q-Learning etc which estimate the value function or Q-function of the current policy with policy based methods like REINFORCE etc. which work by taking gradient of the objective function. The two main constituents in the actor critic algorithm, that is, the actor and critic both have a distinct but significant role to play. The critic, as the name suggests, is used for estimating the action-value function and it updates the **action value function** parameters while the actor updates the **policy parameters** - and this is how actor-critic differs from pure value based approximation methods.

The actor critic algorithms follow an approximate policy gradient, which can be represented using the equations below

$$\Delta_\theta J(\theta) \approx E_{\pi_\theta}[\Delta_\theta log \pi_\theta(s,a) Q_w(s,a)]$$

In this assignment, we have started off with the **Advantage Actor Critic (A2C)** Algorithm and then we implemented an advanced actor critic algorithm as well - **Twin Delayed Deep Deterministic Policy Gradient (TD3)**. We will briefly go through each of these algorithms in the next sections.

1.0.2 A2C

A2C or Advantage Actor Critic algorithm is one of the most fundamental actor-critic algorithm in which the advantage function is used to determine how good an action is in comparison with other actions, at a particular state and on the other hand, the value function which is used essentially tells us the profitability of being in the current state.

The policy gradient in case of A2C can be given as

$$\Delta_\theta J(\theta) = E_{\pi_\theta}[\Delta_\theta log \pi_\theta(s,a) A_w(s,a)]$$

So, in contrast, instead of learning the Q-Values, the critic learns the advantage values and so the overall estimation of an action is not only based on how much better the current action is but how it can be improved further too which has the benefit of reducing the variance in results.

1.0.3 TD3

TD3 or Twin Delayed Deep Deterministic Policy Gradient algorithm is another advanced variant of actor-critic algorithm which in general aims at reducing the overestimation bias as compared to other actor-critic methods like DDPG. It is an Off-Policy algorithm and the algorithm essentially learns two (twin) Q-functions instead of one and we select the smaller one of two to create the targets in Bellman loss function.

This algorithm can be used for environments with continuous action spaces, and to improve upon the stability updates to policy network are lesser in frequency as compared to value network updates this will cause lower variance in value estimates and hence enhanced stability. The main paper for TD3 recommends one policy update for every two Q-function update.

Furthermore, in TD3 clipped noise is added to the selected action during target calculation, so that stable actions can have higher values and thus higher preference. The TD3 pseudocode is given below for reference

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ
with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

Figure 1: TD3 Algorithm

So, to sum up TD3 algorithm adds three major features - clipped double Q learning, delayed policy updates and target policy smoothing by addition of noise.

2 Environment Details

In this section, we will discuss about all the environments used in the assignment along with their properties and other metrics.

2.1 Pendulum

So, starting off with the **Pendulum-v0** environment, In this environment, we have a pendulum which is pivoted at one end and the goal here is to make the pendulum stay upright using a series of left/right motions, which have different values assigned in a particular range. The details can be summarised as follows:

- **Observation** 0 (cos(θ)), 1 (sin(θ)), 2 (θ dot).
- **Actions** Ranges from -2.0 to 2.0
- **Reward** Lowest reward is -16 and highest is 0.
- **Starting state** A random angle between $-\pi to \pi$
- **Episode termination** No specified termination, we can add number of steps or episodes to terminate or can set a predefined value which when average reward reaches, we can terminate

Please see the diagram below, generated from running the *Pendulum-v0* environment locally.

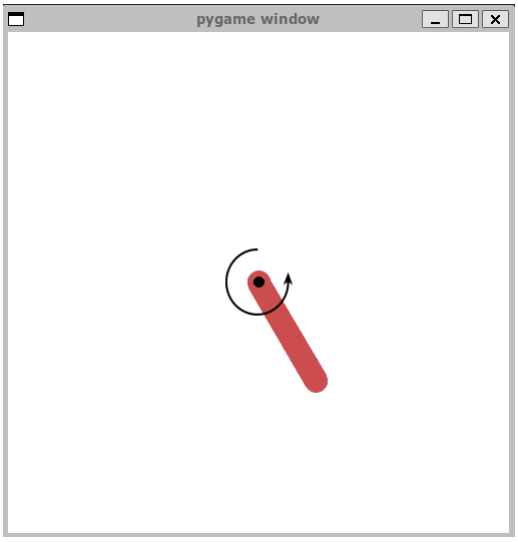

Figure 2: Locally rendered Pendulum-v0 environment

2.2 Lunar Lander

Now, coming to the next environment, we used the second environment as **LunarLander-v2** which is a classic environment in which a simulated rocket ship tries to land on the surface of the moon by maneuvering left and right in the air so that it lands within a specified area in the environment, which is the area between two flags. Please see the screenshot for a clear visualization of the environment. The environment details can be summarized as follows:

- **Observation** 8 in total, including angular velocities

- **Actions** Do nothing, fire left engine, fire right engine, fire main engine
- **Reward** Range from 100 - 140 when trying to land, leg ground touch gives +10, crashing gives -100 and landing gives +100
- **Starting state** Landing pad starts at the origin, (0,0)
- **Episode termination** When the lander crashes or comes to rest position

Please see the diagram below, generated from running the *LunarLander-v2* environment locally.

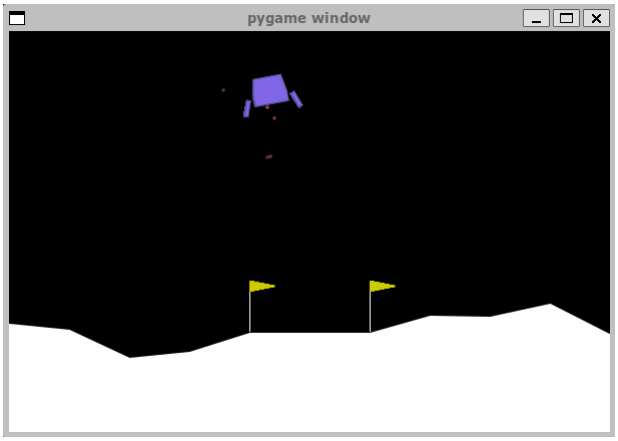

Figure 3: Locally rendered LunarLander-v2 environment

2.3 Bipedal Walker

Finally, the last environment which we have used in this assignment is the **BipedalWalker-v2**, in which we have a two legged two-dimensional structure, and the aim is to make it walk through on a not so smooth terrain. The agent is rewarded based on how far it can go without toppling down on the surface, in which case the episode is considered to be terminated. The environment details can be summarized as below:

- **Observation** 24 in total, ranging from various joint angles to speed and contact flags
- **Actions** 0 (Hip_1), 1 (Knee_1), 2 (Hip_2), 3 (Knee_2)
- **Reward** Rewarded for moving forward, if reaches to the end, can accumulate upto 300 points
- **Starting state** Straight legs and upright in position
- **Episode termination** If the agent reaches to the far right end of the environment or when falls down on the ground

Please see the diagram below, generated from running the *BipedalWalker-v2* environment locally.

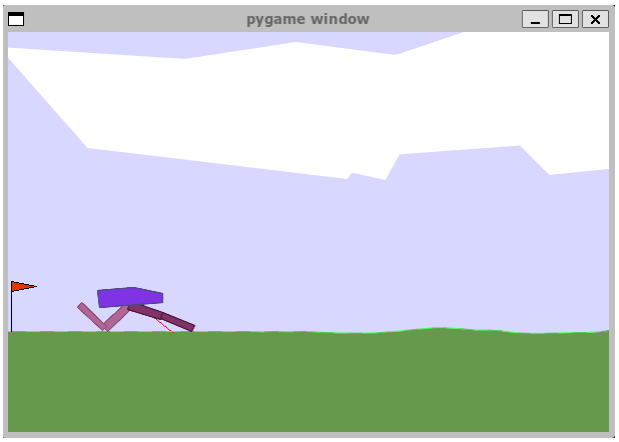

Figure 4: Locally rendered BipedalWalker-v2 environment

3 Results and Analysis

We started with A2C algorithm and then we attempted to implement TD3 algorithm, which was slightly harder to implement as we had to take care of the noise, replay buffer and many more. We are listing down the results which we got using the screenshots below

So, starting with the **A2C** implementation – we ran the algorithm first on the Pendulum environment and ran it for episode range of 500 involving multiple iterations and the cumulative rewards secured in this case was **-278** the graph after completion of training was as follows.

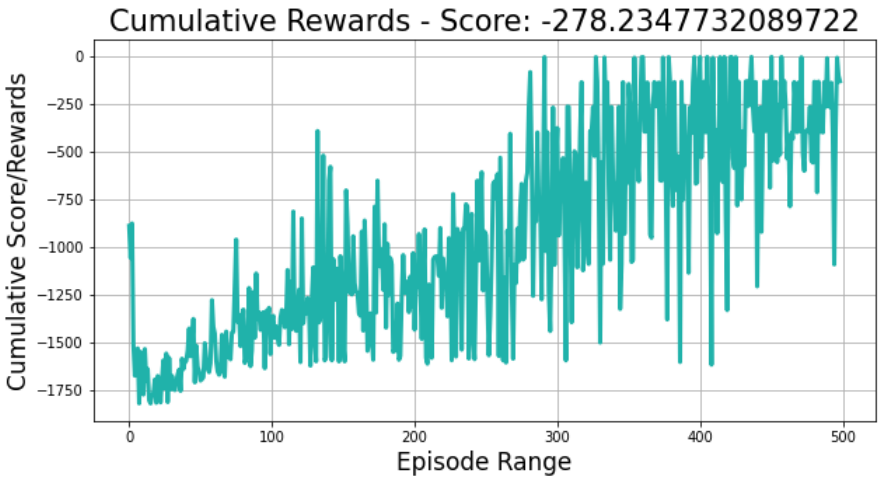

Figure 5: A2C - pendulum training rewards plot

On evaluating, the mean reward was **-0.45** as shown in the plot below

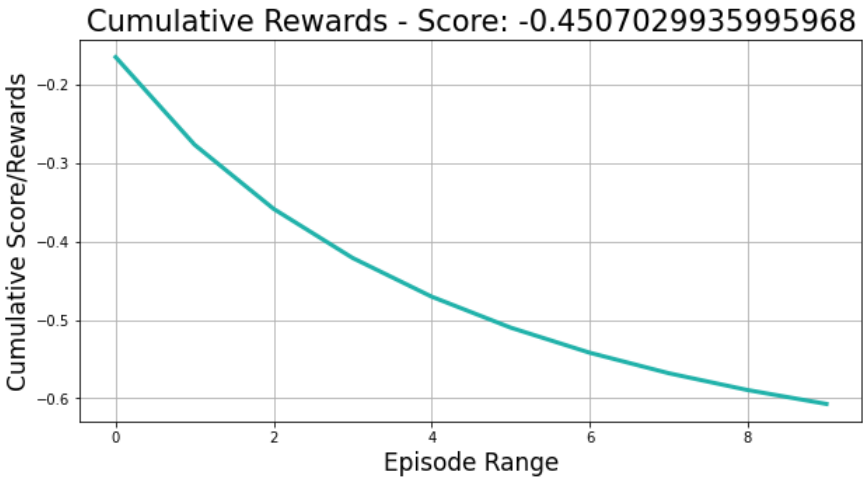

Figure 6: A2C - pendulum evaluation rewards plot

Also, the critic and the actor losses in this case were as follows:

Figure 7: A2C - pendulum actor loss

Figure 8: A2C - pendulum critic loss

As requirement of the part 2 of the assignment, we ran the algorithm in other two environments as well and the results we obtained were as follows.

In case of Lunar Lander environment, the reward metric during training were as follows, reward of **-87.91** was obtained

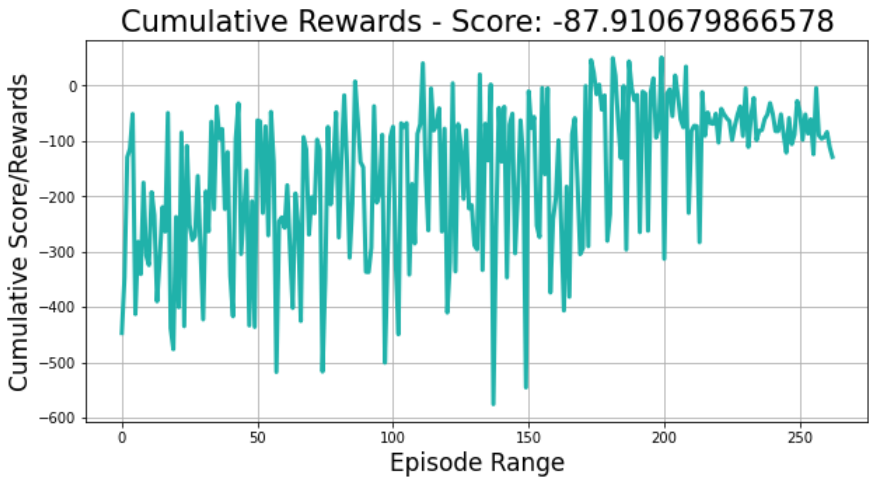

Figure 9: A2C - lunar lander training rewards

and during the evaluation, the mean rewards accumulated were **12.43** as shown in the plot below

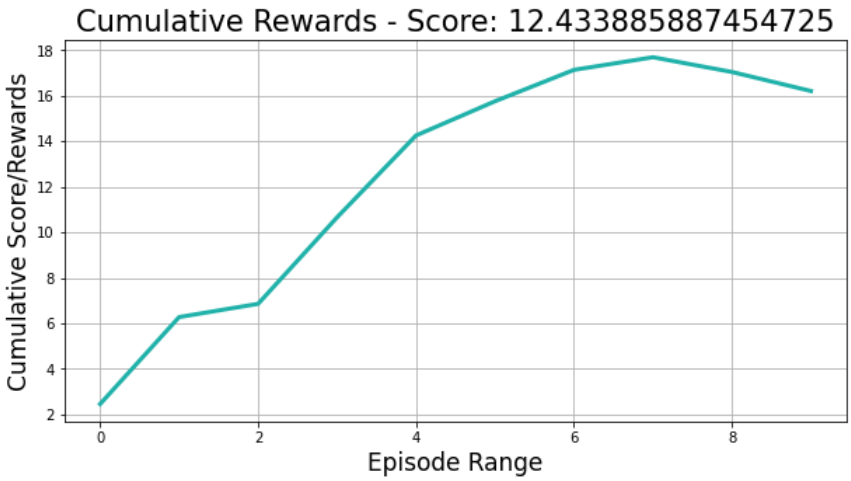

Figure 10: A2C - lunar lander evaluation rewards

The corresponding actor and critic losses are also plotted down below

Figure 11: A2C - lunar lander actor loss

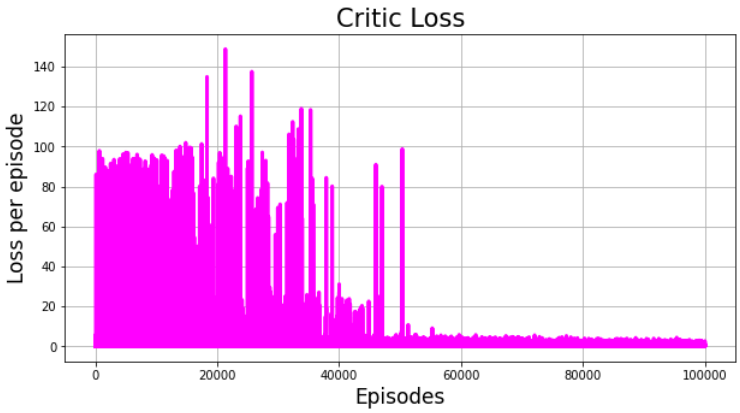

Figure 12: A2C - lunar lander critic loss

Finally, on running the algorithm in a similar manner on Bipedal walker environment, we got following results:

During training, we accumulated a score of **-117.32** the reward metrics remained kind of same in this case.

Figure 13: A2C - bipedal walker training

Upon evaluation over 10 iterations, we got a score of **-0.46** on average as shown in the screenshot below

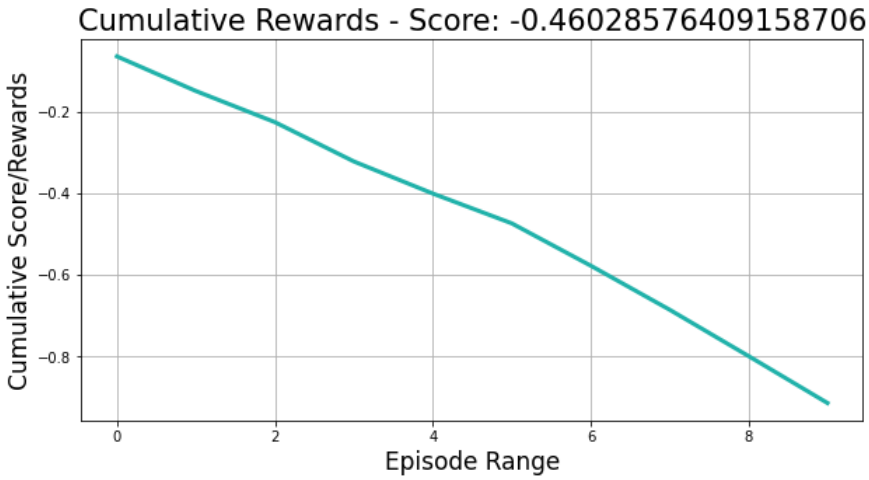

Figure 14: A2C - bipedal walker testing

Now, coming to the extension of the previous algorithm, we tried to implement the TD3 algorithm and tested out different environments here as well and the results are enumerated down below:

First of all, starting with Bipedal walker environment in this case, during training we accumulated rewards of as shown in the graph below

Figure 15: TD3 - bipedal walker training

On evaluating, we got below results - score of as shown in the screenshot below

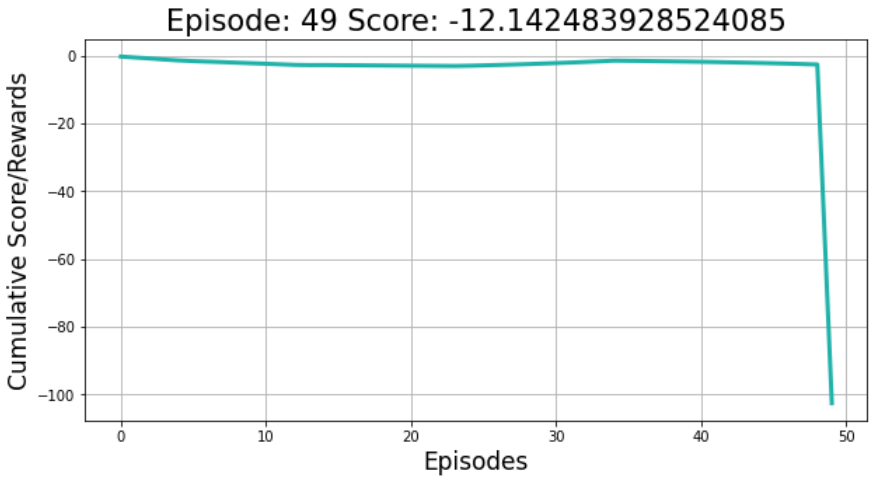

Figure 16: TD3 - bipedal walker testing

We also tested out other two environments, and we got below results - for Lunar Lander environment, during the testing, we accumulated a score of **-444** as shown in the screenshot below

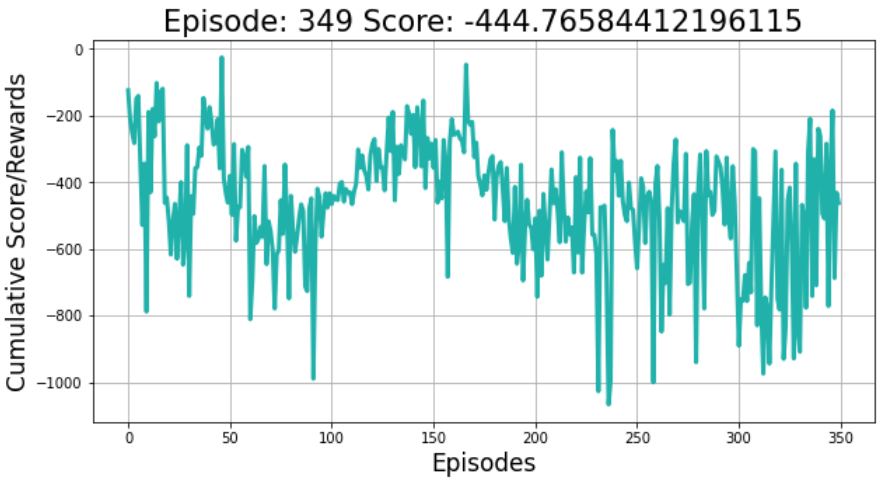

Figure 17: TD3 - lunar lander training

On testing, the results were as shown below

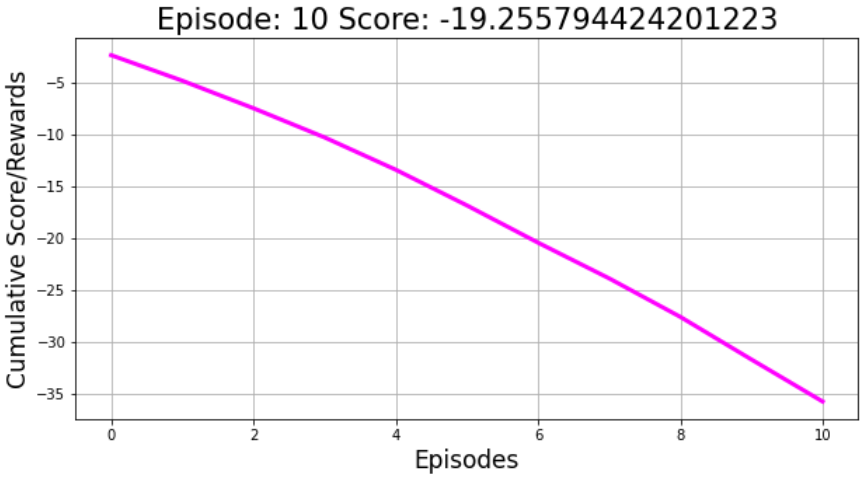

Figure 18: TD3 - lunar lander testing

Finally, coming to the pendulum environment, on training we got a score of **-1359** as shown in the screenshot below

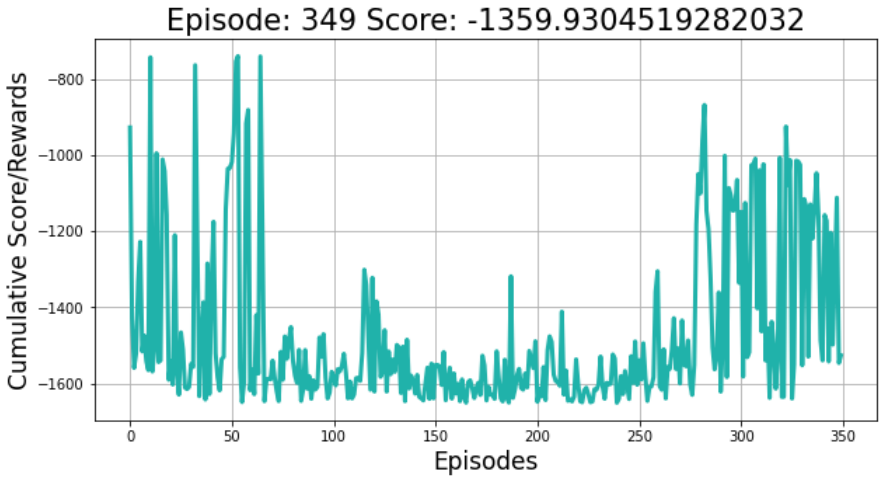

Figure 19: TD3 - pendulum training

On testing, the results were as shown below

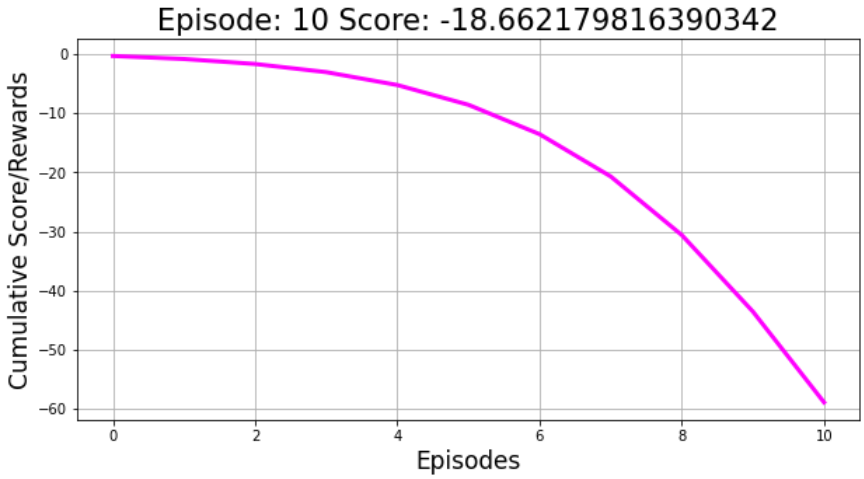

Figure 20: TD3 - pendulum testing

So, overall this was a slightly challenging assignment to do but we had a lot to learn – there were many environment specific tweaking required both in terms of hyperparameters and rewards, but the same was not consistent across environments, which was challenging to manage at times.

4 Contributions

Team Member(s)	Assignment Part	Contribution
Naman, Shreyans	A2C	50% and 50%
Naman, Shreyans	TD3	50% and 50%
Naman, Shreyans	Documentation	50% and 50%

References

[1] Lecture Slides

[2] Bi-Pedal Walker Environment https://github.com/openai/gym/wiki/BipedalWalker-v2

[3] Lunar Lander Environment https://github.com/openai/gym/wiki/Leaderboard#lunarlander-v2

[4] Pendulum https://github.com/openai/gym/wiki/Pendulum-v1

[5] Mountain car problem https://en.wikipedia.org/wiki/Mountain_car_problem

[6] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[7] Fujimoto S, et al, "Addressing Function Approximation Error in Actor-Critic Methods" https://arxiv.org/pdf/1802.09477.pdf

[8] Twin Delayed DDPG https://spinningup.openai.com/en/latest/algorithms/td3.html

[9] TD3: Learning To Run With AI https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93