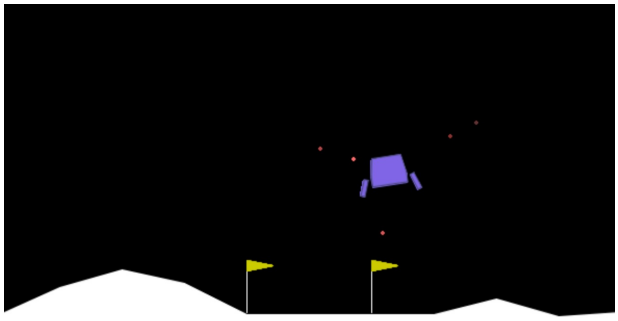
Lunar Lander using Open AI Gym

Sarthak Garg(122101035) and Shubhan Mehrotra(122101037)

27 November 2023

1 Introduction

The project is to create a model that will allow a lunar lander to land at a given location. The lander can perform 4 actions. It can fire the three engines, main, and left and right or do no action. The observation space is defined as an 8 dimensional vector: the coordinates of the lander in x & y, its linear velocities in x & y, its angle, its angular velocity, and two booleans that represent whether each leg is in contact with the ground or not. The lander has to successfully land in the given area. The 2d space acts as our entire space in which the lander is permissible to be. Anywhere outside and the episode is terminated. The lander has to be trained to successfully land in the defined zone. There are large penalties for touching the moon outside the defined are (i.e. crashing). There are penalties for firing any of its three engines. There are additional rewards fro each leg touching ground and additional penalty for moving away from the defined zone. These promote the algorithm to use the least amount of fuel to reach the area and also means that the lander will land with both its leg touching the ground i.e. in a stable state. An example state if the system is given below which show the position of the lander and the thrusters being fired along with the moon and the defined landing zone.



This agent can be trained using Deep Q-Learning and will strengthen the concepts of Deep Q-Learning. The final rewards after each episode will motivate the agent to find the most effective solution and it will become more and more efficient as more episodes occur.

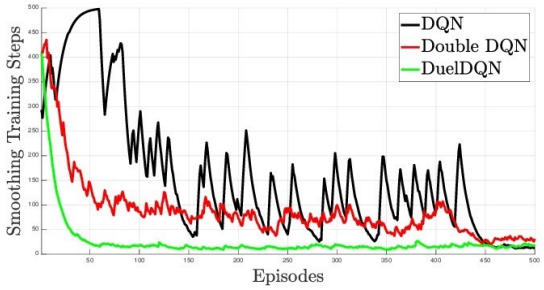
2 Objectves

1. Make a lander which can land successfully on the moon.
2. To optimize fuel usage and accurately land in predefined landing zone.
3. Land with both legs touching the surface to achieve stability.
4. To remain aligned with the final landing zone.

3 Motivation

It serves the purpose of simulating lunar lander which mirrors the real world challenges it faces

in robotic autonomous systems used for space exploration. Landing a space with minimal fuel usage thus optimising resource consumption. It also helps us to achieve precise positioning in the landing area. Working on this project with DQN allows for a hands on understanding of how Q-learning is extended to Deep Neural Networks .Being able to solve this problem in 2D gives us a better understanding on how a machine learns.



D3QN vs DQn

4 State-of-the-art

State of the art for solving the lunar lander involves the use of Dueling Double Deep Q-Network (D3QN) which is also a modern technique used in lunar lander simulations. D3QN replaces the shared Q-network used for both action selection and Q-value estimation in DQN with two separate Q-networks. D3QN is an improvement over DQN and the major components of its approach are as follows:

1. **Dueling Architecture:** Splitting the network into two paths to separately assess start value and action advantages

2. **Double Q-learning :**Using dual networks reduces overestimation bias by separating action selection and value estimation.

3. **Experience Replay :** Stores samples in replay buffer and samples past experiences to enhance learning efficiency and break temporal correlations

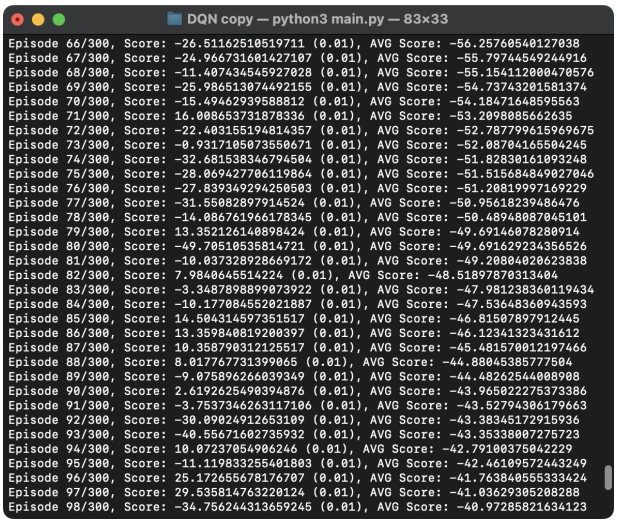
4. **Target Networks :** To ensure a more stable training process and provide consistent target Q-values, separate networks are maintained.

The overestimation of Q-values by DQN can result in sub optimal policies and a slower convergence rate, this is not feasible when dealing with complex or noisy environments. This can be countered with D3QN which improves learning stability which leads to faster convergence and more consistent learning.

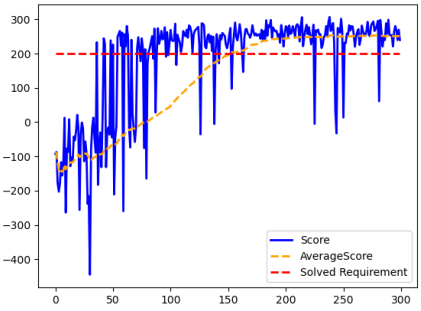
5 Methodology/Approach

To being with the project, we need to define an environment. For this project we will be using a predefined environment in the package gym which is a Box2d based environment with a lunar lander trying to land on the moon. Now we need to train the AI to be able to solve any given state to it in this environment. We do this by generating a random state that is given by the package itself which is the lander starting at the top centre of the viewport with a random initial force applied to its centre of mass. We let the agent run over for the given number of episodes in our case which was 300. We let the agent interact with the environment until each episode is done. The Episode terminated if the lander crashes (touches the surface), the lander gets out of the defined environment i.e. x coordinate greater than 1 or the lander stops moving. After each episode ends the environment resets itself. Within the loop the agent selects actions using its exploration-exploitation policy, where it either explores with a random action or exploits by choosing the action with the highest Q-value. The environment is stepped forward with the chosen action, and the resulting state, reward, and done status are obtained. The experience tuple (state, action, reward, new_state, done) is stored in the replay buffer. The overall training happens as follows. If there are enough samples in the replay buffer, a batch of experiences is sampled. The online Q-network is updated using the DQN algorithm. The target Q-values are calculated using the Bellman equation. The loss is calculated as the mean squared error between the

predicted Q-values and the target Q-values. The network is trained using the current batch. The exploration parameter (epsilon) is decayed over time. The target Q-network is periodically updated with the weights of the online Q-network to enhance stability. There is also a part of performance tracking that happens. Scores for each episode are recorded, where score is the cumulative reward obtained during the episode. The average score over the last 100 episodes is calculated and it is also recorded. The current episode's score, epsilon value, and average score are printed for monitoring the training progress. The rewards have been defined as the reward for moving from the top of the screen to the landing pad and coming to rest is about 100-140 points. If the lander moves away from the landing pad, it loses reward. If the lander crashes, it receives an additional -100 points. If it comes to rest, it receives an additional +100 points. Each leg with ground contact is +10 points. Firing the main engine is -0.3 points each frame. Firing the side engine is -0.03 points each frame. Solved is 200 points. We have taken the final solution being acceptable when the average score reaches 200 and the current score is above 250.



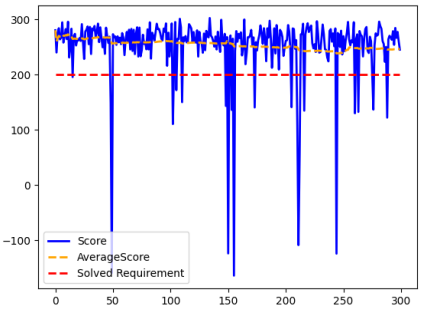
Testing Process



Training Score Graph

6 Results and Discussion

Observing the total reward for the episodes and the penalties we can see that even while training the AI, after the first hundred episodes the AI can see that the reward is highly negative. Here a negative reward is given when the lander crashes or moves away from the landing zone or uses any of its engines. As we move forward the negative reward decreases and we see that the reward starts going positive we also see that the average reward over the last 100 episodes is increasing at a steady state. We see that after the first 150 episodes we reach the solved requirement and then we are going further just to improve the model even further. When viewing the testing graph over 300 episodes we see that the average score remains close to 250 over the entirety of the dataset and there are only a few times where the score has gone below the solved threshold.



Testing Score Graph

7 Conclusion

Observing the performance of the Al after training for 300 episodes and testing for the same we can see that we have finally obtained trained Al that is capable of successfully landing a lunar lander on the moon in a given landing zone while using the least fuel possible. The Al successfully uses the fuel in an optimal manner and lands on the correct area of the moon while maintaining stability by keeping both of its keep touching the ground.

8 Future Prospects

Using this 2D Lunar Lander and training a model where the lander tries minimising its air time and precise positioning in the landing area can be set as a base for creating more realistic 3D simulations that could provide agents with diverse challenges and opportunities to learn. The model can learn similar complex tasks like :

1. Acceleration Control: Training the model to minimize acceleration during landing by fine-tuning thrust and engine power to gradually decrease velocity for a safe descent.

2. G-Meter Management: It is crucial to ensure the safety of the craft and its occupants by minimizing the G-forces experienced during landing. To achieve this, the model can regulate the descent speed and the rate of deceleration to keep the G-forces within safe limits.

3. Soft Touchdown: To achieve a gentle contact with the ground, it is necessary to control the descent velocity. The model can be trained to gradually decrease the vertical speed by adjusting the engine thrust or control surfaces. This will facilitate a controlled and soft touchdown.

All these changes will make the model more realistic and ready to take on more challenges in the real world.

Simulated environments frequently lack specific complexities found in the real world, including sensor noise, environmental uncertainties, and hardware constraints. This disparity between simulation and reality can pose challenges when attempting to directly apply learned behaviors to physical systems. This model can only act as a stepping stone to learn better methods such as Dueling Double Deep Q-Networks which will lead us to train lunar lander models. The algorithm is used to program a lunar lander land while efficiently managing its resources and landing in the designated region. These learnings have been applied to a variety of other reinforcement learning problems, and they have helped to improve the performance of these algorithms.

9 References

1. Gao, J.; Ye, W.; Guo, J.; Li, Z. Deep Reinforcement Learning for Indoor Mobile Robot Path Planning. Sensors 2020, 20, 5493. https://doi.org/10.3390/s20195493

2. Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. https://arxiv.org/abs/1710.10044

3. Castro, P. S. et al. (2018). Distributional Dueling Q-learning. arXiv preprint arXiv:1710.10044. https://www.nature.com/articles/nature14236