Reinforcement Learning for Lunar Lander

Ahmad Aldaher, Ahmad Hamdan , Ghadeer Issa

May 1, 2024

I Introduction

Reinforcement learning (RL) is a pivotal method in artificial intelligence that trains algorithms using a system of rewards and penalties. It enables machines to make decisions and learn optimal behaviors within a dynamic environment based solely on feedback. This project leverages RL to tackle the Lunar Lander problem provided by OpenAI's Gym interface, a popular testbed for developing and comparing reinforcement learning algorithms.

The Lunar Lander task simulates the challenge of autonomously landing a spacecraft on the moon. The agent must learn to control the lander by firing its thrusters at appropriate intervals to touch down at a designated spot without crashing. Success in this environment requires mastering a balance between safe landing techniques and fuel efficiency, encapsulating the complexities typical in real-world control tasks.

Our investigation begins by applying Q-learning with discretization of the state space to establish a baseline performance in the environment. This approach converts the continuous state variables of the Lunar Lander into discrete states, making traditional tabular Q-learning feasible. Following this, we extend our exploration to more sophisticated RL algorithms such as Deep Q-Networks (DQN), Double DQN, and Advantage Actor-Critic (A2C). These algorithms represent cutting-edge techniques in handling high-dimensional state spaces and balancing exploration with exploitation.

Moreover, the project seeks to explore the robustness and adaptability of these algorithms under modified conditions. First, by altering the reward function of the default environment, we aim to study how changes in reward structures influence learning outcomes and behavior. Subsequently, the introduction of an obstacle in the landing zone tests the algorithms' ability to adapt to new challenges and optimize under altered environmental dynamics.

This report details the methodology, experiments, and results of applying Q-learning, DQN, Double DQN, and A2C to the Lunar Lander environment, both in its default configuration and under modified conditions. By analyzing and comparing the performance of these algorithms, this study aims to contribute valuable insights into the applicability and efficiency of various RL approaches in complex control tasks.

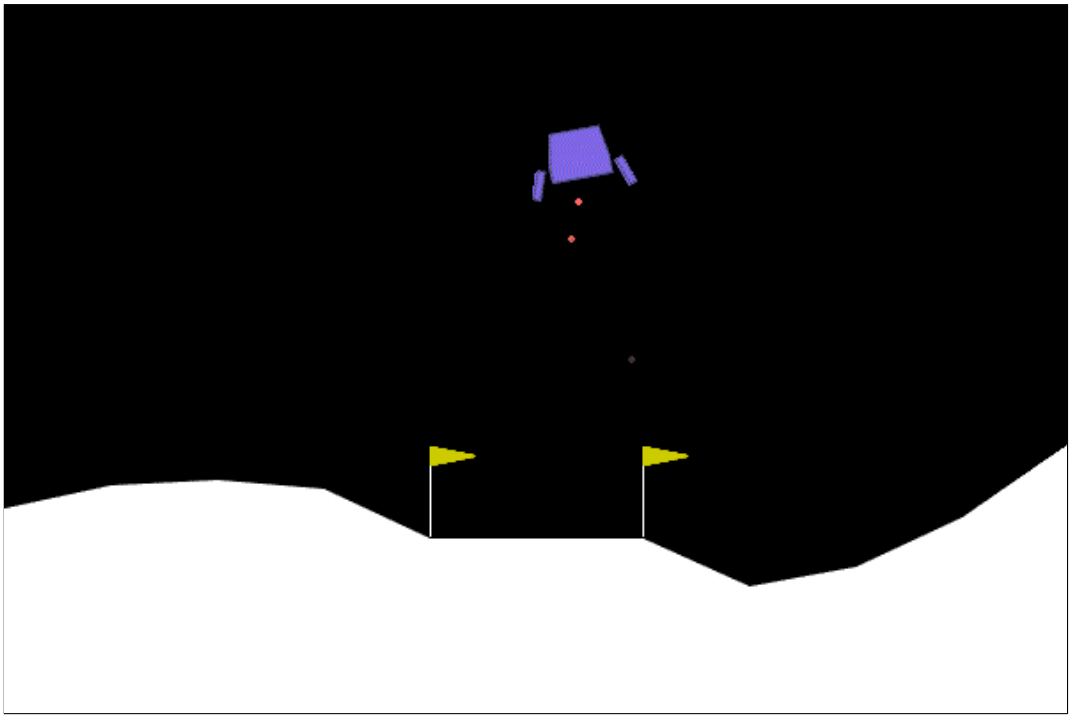

Fig. 1. Lunar Lander

II Lunar Lander Environment

The Lunar Lander environment simulates the scenario of landing an autonomous spacecraft on the moon. The goal is for the agent to control the lander and safely land it on a landing pad located at coordinates (0,0). The lander starts at the top of the screen with a random initial position and velocity. The agent controls the lander by firing the thrusters, which influence its horizontal and vertical velocity.

TABLE I
Lunar Lnader Parameters

Action Space	Discrete(4)
Observation Shape	(8,)
Observation High	[1.5 1.5 5.0 5.0 3.14 5.0 1.0 1.0]
Observation Low	[-1.5 -1.5 -5.0 -5.0 -3.14 -5.0 -0.0 -0.0]

Action Space: There are four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

Observation Space: The state is an 8-dimensional vector: the coordinates of the lander in x and y, its linear velocities in x and y, its angle, its angular velocity, and two booleans that represent whether each leg is in contact with the ground or not.

Rewards: 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.

A. Tested Scenarios

we used three scenarios in our work to test the algorithm:

1. **The Default Lunar Lander Environment** we tested all the algorithms on the default lunar lander environment (Fig1) from the gym library.

2. **Modified Reward**: we modified reward function for the lunar to get better performance. We increased the penalty on the position and decreased it on the velocity in order to make the lander land faster, also we added a penalty when the reward is less than -400 to prevent the lander form unnecessary hovering over the landing spot and to land faster. Also we increased the reward for each leg contact with land to 20.

3. **Adding Obstacle into The Environment**: We added a rectangle static obstacle in the way of the lander and the lander will crash and lose the game if it collided with obstacle and get a penalty of -100. The propose of adding this obstacle is to see if the lander will manage to avoid colliding with obstacle during the landing.

III Q Learning

Q-learning is a model-free reinforcement learning algorithm that seeks to learn the optimal action-selection policy using a Q-table, a matrix-like structure where rows represent states and columns represent possible actions. The central goal of Q-learning is to learn the values of this Q-table, which effectively stores the expected future rewards for a state-action pair.

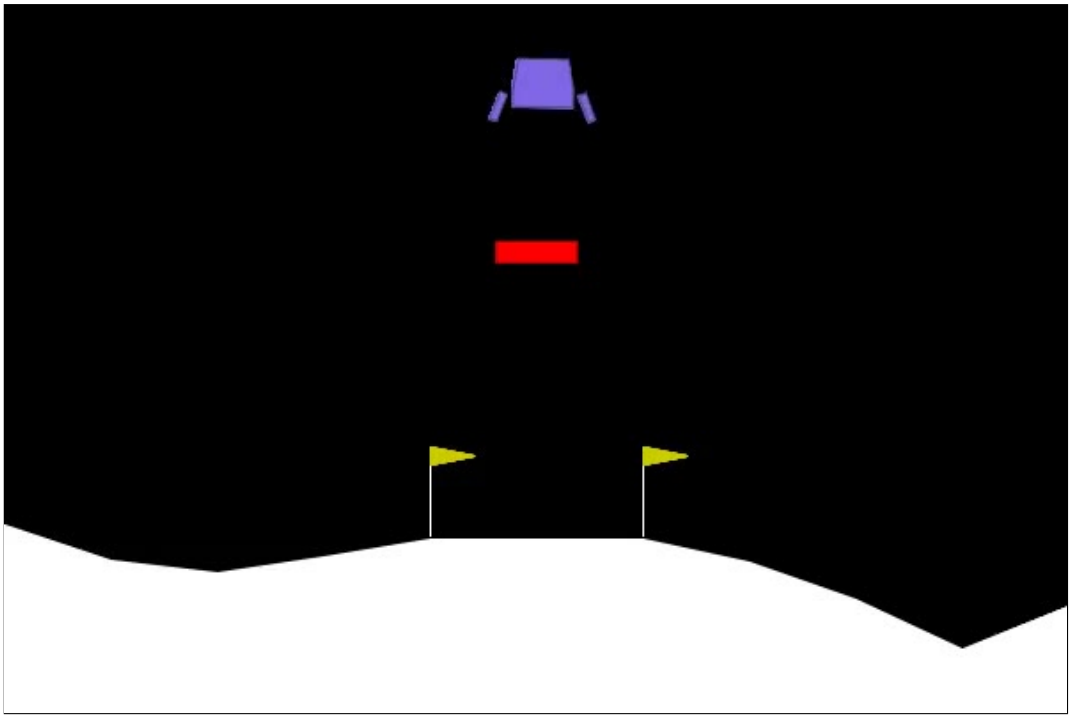

Fig. 2. Lunar Lander with obstacle

How Q-learning Works The process begins by initializing the Q-table with arbitrary values, and as the agent interacts with the environment, these values are updated based on the equation:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

where:

- $Q(s,a)$ is the current Q-value of being in state s and taking action a.
- α is the learning rate, which determines to what extent newly acquired information overrides old information.
- r is the reward received after taking action a in state s.
- γ is the discount factor, used to balance immediate and future rewards.
- $\max_{a'} Q(s',a')$ the estimated maximum future reward achievable from the next state s'.

The agent uses this Q-table as a reference to choose actions by selecting the one with the highest value (highest expected reward) for each state. This is typically done using a policy like

ε-greedy, where most of the time the best action is chosen but with a small probability ε, a random action is selected to encourage exploration of the state space.

In our case for the Lunar Lander environment we have a continuous state so to be able to apply Q learning we need to discretize the state. we divided the range for the contious state between high level and low level into 20 sections and we keep the discrete two state related to the legs contact the same. Thus we have the following shape for the Q-Tabel:

$$size(Q - Table) = (20, 20, 20, 20, 20, 20, 1, 1, 4)$$

We tested Q-Table learning on the Lunar lander environment and 50000 episodes we found out the lander still fail in landing safely on the lunar object, and this due to the discretization. If went a better results when need to use a more fine resolution and this lead more need for memory and increasing in the time to update the Q-Table in each step. Fig(11) shows the rewards vs the episode when using Q learning.

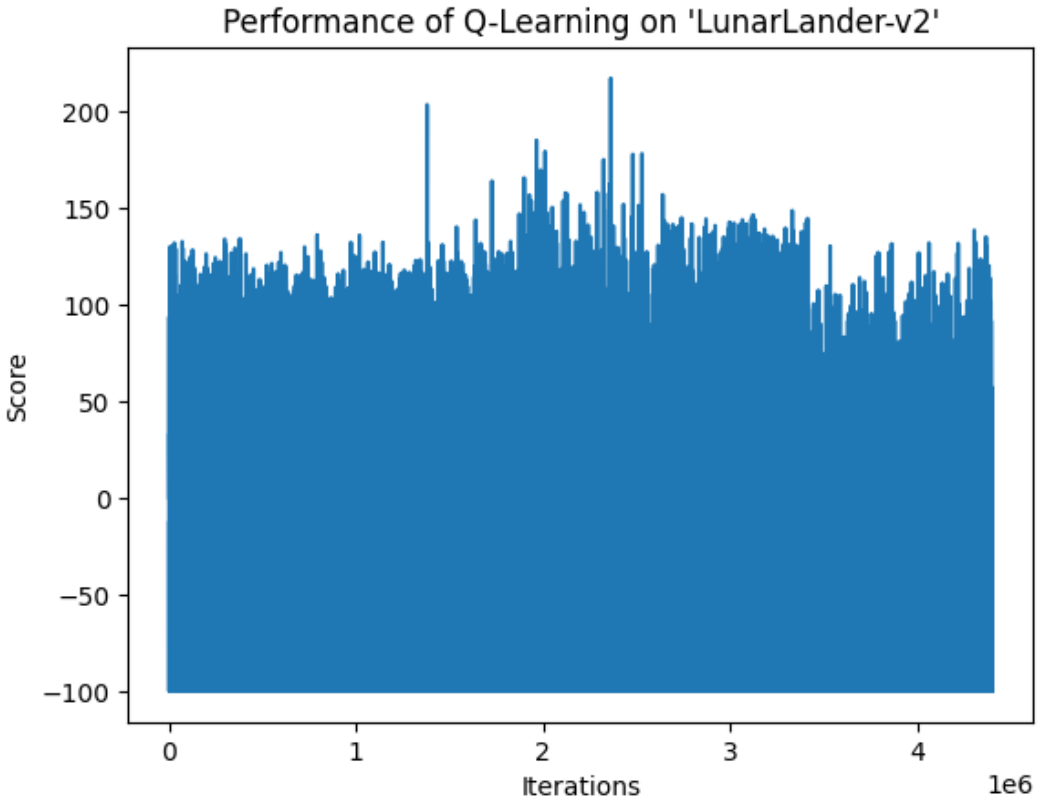

Fig. 3. Q Learning (reward vs episode)

In general Q-learning faces several key limitations, particularly in complex environments.

It doesn't scale well to large or continuous state spaces due to the exponential growth of its Q-table. This method is also slow to converge, memory-intensive, and requires discretization of continuous states, which can lead to performance degradation. Therefor we need to switch to function approximating to find our Q-function $Q(s, a)$ and use neuronal networks based algorithm like DQN.

IV Deep Q Networks (DQN)

Deep Q-Networks (DQN) revolutionized the field of reinforcement learning by successfully integrating neural networks with Q-learning. Introduced by researchers at DeepMind in 2015, DQN addresses many of the limitations associated with traditional Q-learning, particularly its scalability and generalization issues in environments with high-dimensional state spaces.

DQN extends Q-learning by using a deep neural network to approximate the Q-value function. Instead of maintaining a Q-table, DQN learns the optimal policy by training a neural network to predict Q-values for all possible actions given a state. This approach effectively handles environments with large or continuous state spaces, where traditional Q-learning would be impractical.

The neural network in DQN takes the state of the environment as input and outputs a Q-value for each action. Training involves updating the network's weights to minimize the loss between the predicted Q-values and the target Q-values, which are computed using the Bellman equation:

$$Target \quad Q_{value} = r + \gamma \max_{a'} Q(s', a';\ \theta^-) \tag{2}$$

where r is the reward , γ is the discount factor. $\max_{a'} Q(s', a';\ \theta^-)$ epresents the maximum predicted Q-value for the next state s', using the weights of a target network θ^- which are periodically updated with the weights of the main network to stabilize training.

Key Features of DQN:

- **Experience Replay**: DQN utilizes a technique known as experience replay, where transitions (state, action, reward, next state) are stored in a replay buffer. Random mini-batches from this buffer are used to train the network, breaking the correlation between consecutive learning samples and thus stabilizing the learning process.

- **Fixed Q-targets**: To further stabilize training, DQN uses a separate target network with fixed

parameters to calculate the target Q-values. The weights of this target network are updated less frequently than the primary network, reducing the risk of harmful feedback loops during training.

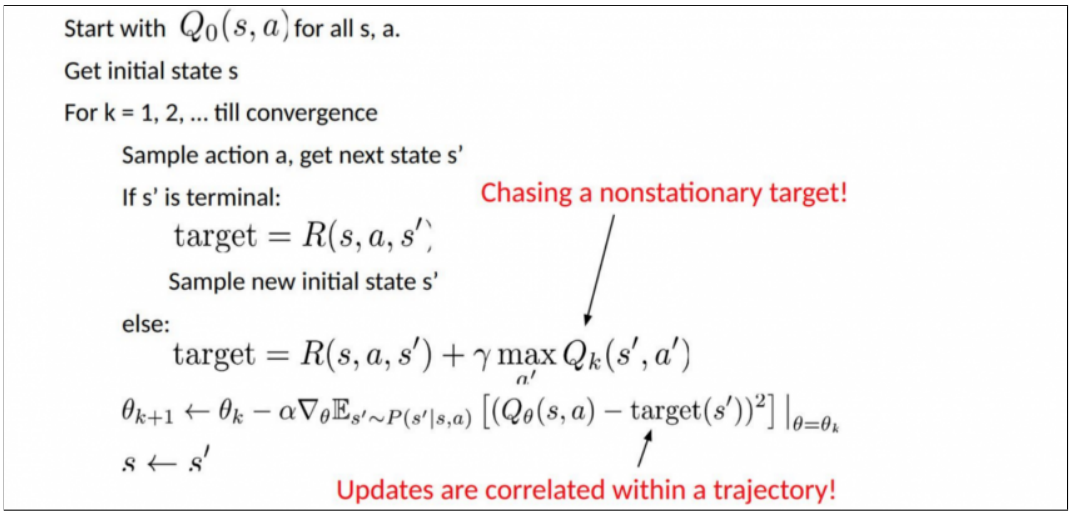

Fig. 4. DQN algorithm

Advantages of DQN:

- Handling of High-Dimensional Spaces: DQN can manage environments with large state spaces, making it suitable for complex tasks like video games or robotics.
- Generalization: Unlike traditional Q-learning, DQN can generalize from seen states to unseen ones, learning efficient representations through its neural network.
- Stability: Techniques like experience replay and fixed Q-targets contribute to the stability of learning, making DQN more reliable than standard Q-learning.

Limitations: While DQN represents a significant advancement in reinforcement learning, it is not without limitations:

- It can be computationally expensive and requires significant amounts of data.
- DQN may overestimate Q-values due to the max operation in the Bellman equation, although variants such as Double DQN help mitigate this issue.
- DQN is still prone to instability in certain environments, especially when the action space is also continuous.

A. Performance Analysis of DQN Lunar Lander

We tested the DQN algorithm the lunar lander environment and we got the following results:

1. **Default Lunar Lander**: we trained our DQN model in the default lunar lander environment. (Fig(5) shows the change in the rewards during the leaning)

2. **Modified Reward**: (Fig(6) shows the change in the rewards during the leaning in the modified reward scenario)

3. **Added obstacle** :(Fig(7) shows the change in the rewards during the leaning after adding an obstacle in the way of the launder.

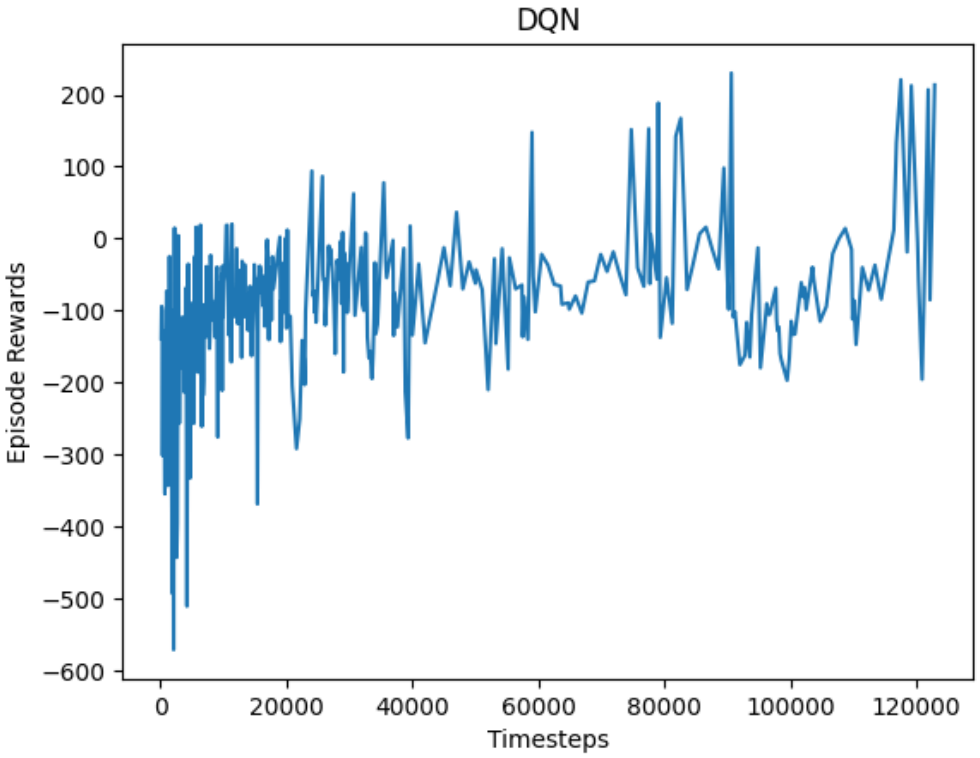

Fig. 5. DQN for default lunar lander

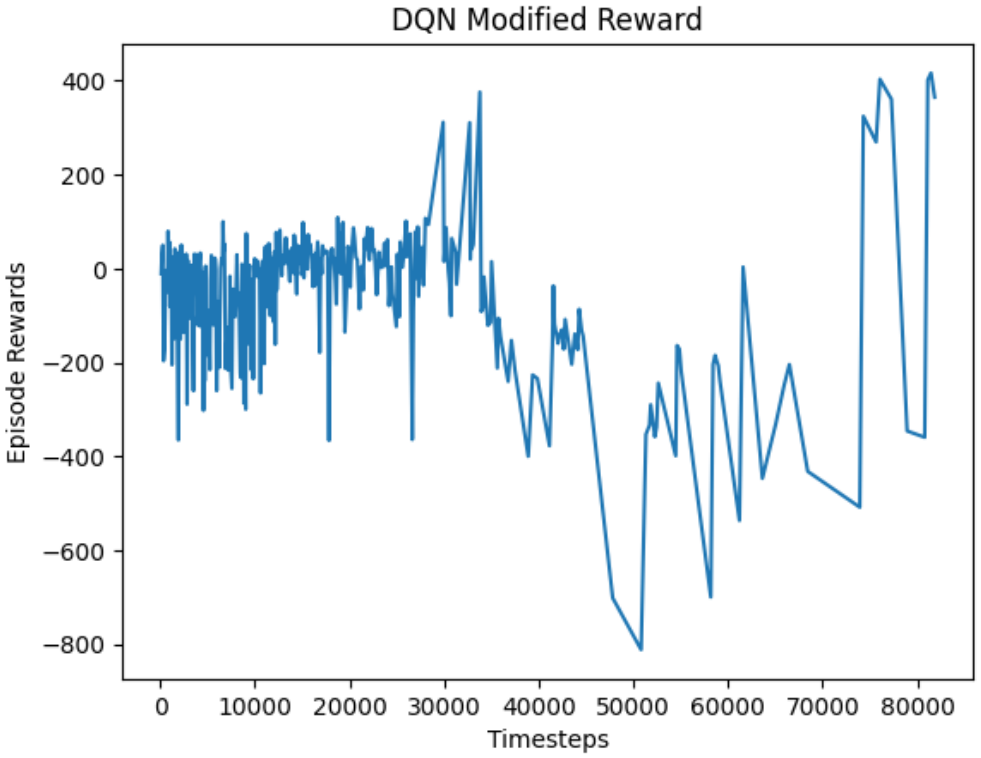

Fig. 6. DQN after modifying the reward function

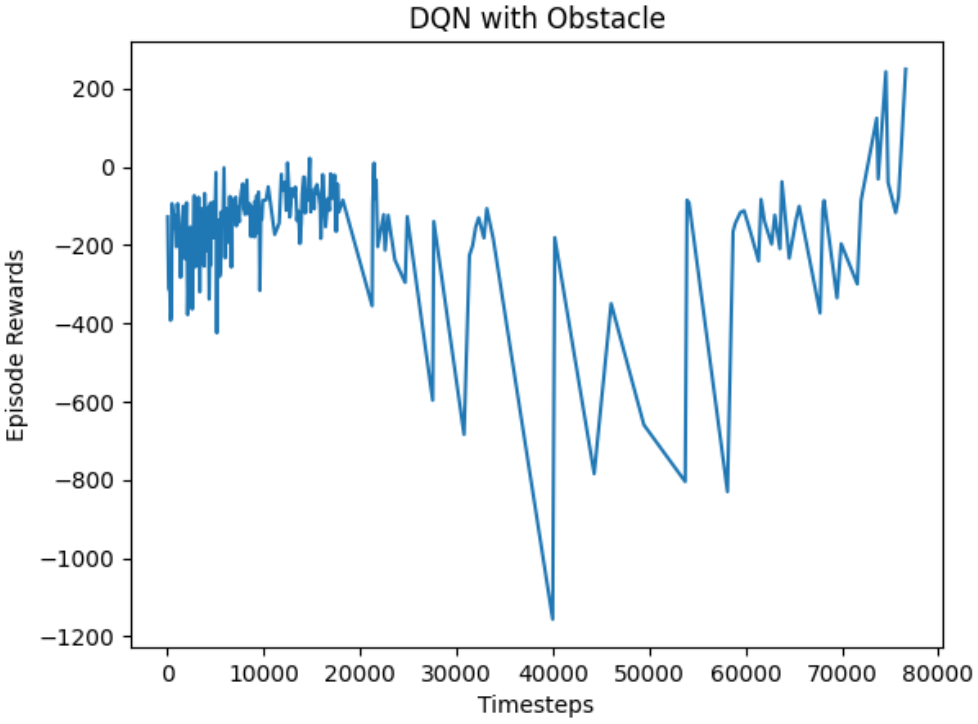

Fig. 7. DQN after adding an obstacle

And the lunar succeed in landing safely in all of the three scenarios. From the figures (5,6,7) we notice from our results the model suffer from fluctuations and there is instability in the learning process, and the reason for this is that the network try to track a moving target. This instability stems largely from the inherent design of how Q-values are updated and estimated within the network.

V Double DQN

Double Deep Q-Networks (DDQN) enhance the traditional Deep Q-Network (DQN) approach by mitigating the overestimation of action values. DDQN employs two neural networks to separate the action selection process from action evaluation:

- **Action Selection:** The online network selects what is perceived as the best action.
- **Action Evaluation:** The separate target network evaluates the action's value, aiming for a more stable and reliable estimation.

This structure helps stabilize learning and leads to more effective policy development when compared to the previous method (DQN)

A. Why Use DDQN for Lunar Lander?

DDQN offers significant benefits for the Lunar Lander problem:

- **Stability:** The separation of networks stabilizes learning updates, crucial in high-variance environments like Lunar Lander.
- **Reduced Overestimations:** DDQN helps guard against the bias of overestimating Q-values, facilitating more accurate learning outcomes.
- **Efficiency:** By reliably estimating action values, DDQN can learn optimal policies more efficiently, optimizing training time and resource usage.

B. Network Design

Construct two neural networks with identical architecture for the online and target networks, typically composed of fully connected layers.

C. Training Process

Training involves several key strategies:

- **Exploration vs. Exploitation:** Use an ε-greedy strategy for balancing between exploring new actions and exploiting known ones.
- **Experience Replay:** Store past experiences in a replay buffer to decouple consecutive learning updates.
- **Periodic Updates:** Regularly synchronize the target network with the online network to ensure stability.

D. Learning Algorithm

Update the online network using the temporal difference (TD) error, which is based on the Bellman equation:

$$Q_{\text{target}} = r + \gamma \cdot Q(s', \arg\max_{a'} Q(s', a';\, \theta);\, \theta^{-}) \tag{3}$$

where:

- r is the reward,
- γ is the discount factor,
- θ represents the online network parameters,
- θ^{-} represents the target network parameters.

E. Performance Analysis of DDQN Lunar Lander

We tested the DDQN algorithm with the lunar lander environment and we got the following results:

1. **Default Lunar Lander**: we trained our DDQN model in the basic lunar lander environment. (Fig(8) shows the change in the rewards during the leaning)
2. **Modified Reward**: (Fig(9) shows the change in the rewards during the leaning in the modified reward scenario)

3. **Added obstacle** :(Fig(10) shows the change in the rewards during the learning after adding an obstacle in the way of the launder.

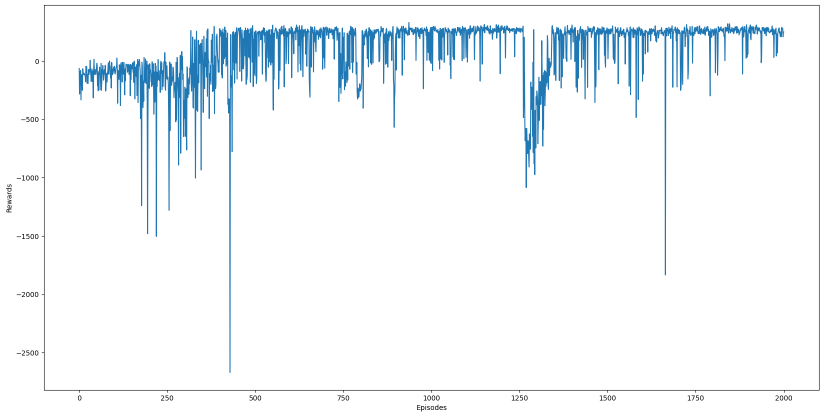

Fig. 8. DDQN for default lunar lander

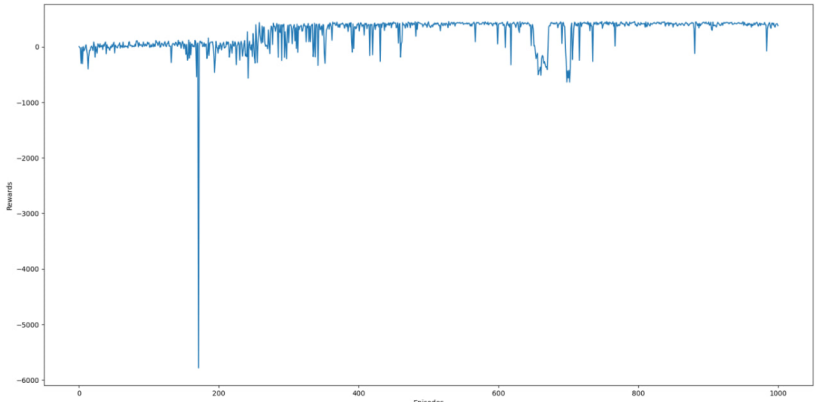

Fig. 9. DDQN after modifying the rewards

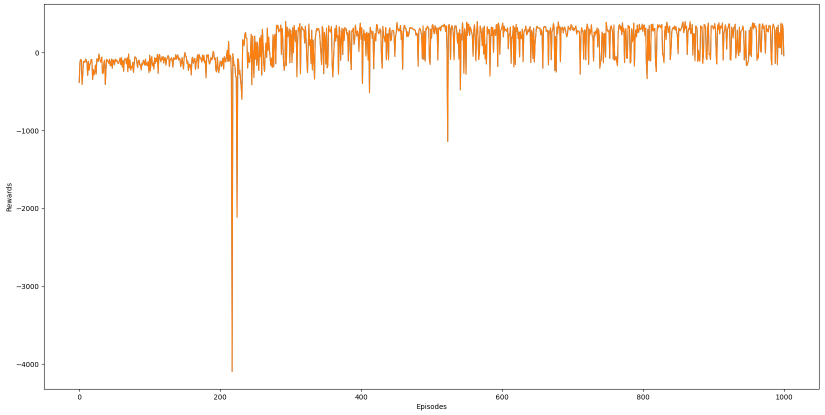

Fig. 10. DDQN after adding the obstacle

F. Conclusion

DDQN offers a robust method for addressing the challenges posed by the Lunar Lander problem, effectively improving upon simpler reinforcement learning algorithms through more accurate value estimation and policy development. This showcases the practical application of advanced reinforcement learning techniques in complex scenarios.

VI SAC

Implementation of the SAC Algorithm for Lunar Lander Environment

A. Introduction

Soft Actor-Critic (SAC) is an advanced reinforcement learning algorithm that optimizes a stochastic policy in an off-policy manner. It combines the benefits of Actor-Critic methods with the robustness of maximum entropy techniques to achieve high sample efficiency, robustness, and effective exploration capabilities by maximizing a trade-off between expected return and entropy, a measure of randomness in the policy.

B. SAC Algorithm Structure

The SAC algorithm comprises several critical components, each serving an essential role in the learning process:

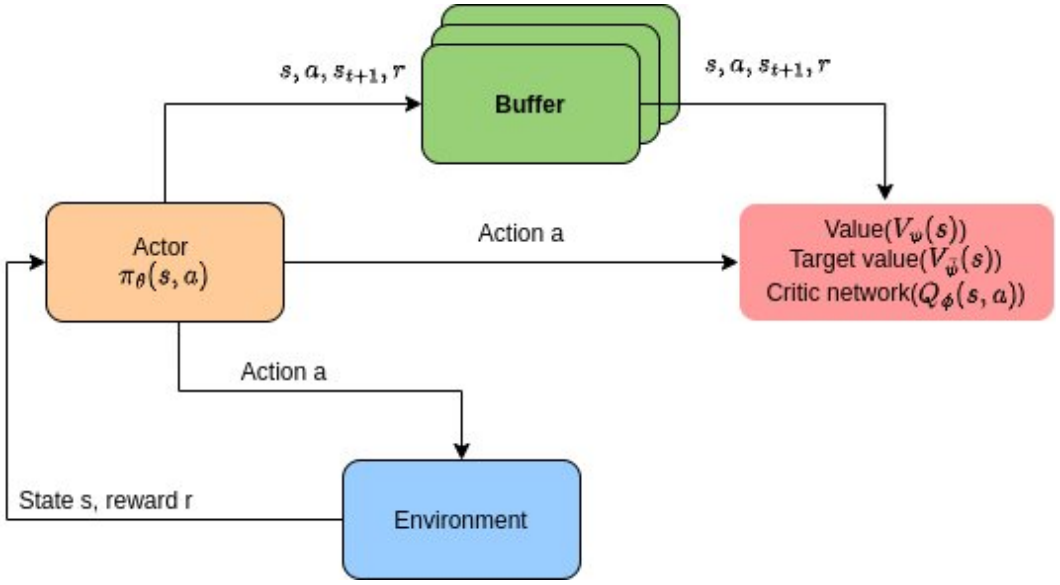

Fig. 11. SAC Algorithm

1. **Actor Network**: This network proposes actions given states. Unlike traditional Actor-Critic methods, where the actor outputs a deterministic action, SAC's actor network outputs a probability distribution over possible actions, rendering the policy stochastic.

2. **Critic Networks**: SAC employs two critic networks, along with two corresponding target networks, which help in mitigating overestimation bias of action values. These networks estimate the Q-values for state-action pairs, providing feedback to the actor network about the action quality.

3. **Entropy Term**: The inclusion of an entropy term in the SAC objective encourages the policy to explore by maximizing the entropy alongside rewards. This leads to more exploratory and robust policy learning, particularly beneficial in environments with many local optima.

4. **Target Networks**: These are slowly updated averages of their corresponding critic networks. They stabilize training by providing consistent, older Q-value estimates for temporal difference error calculations.

5. **Replay Buffer**: SAC utilizes a replay buffer to store past experiences, enabling learning from a diverse range of states and actions and decoupling the temporal correlation of experiences, which promotes more stable and efficient learning.

C. Performance Analysis in Lunar Lander

This section details the performance of the SAC algorithm under three different experimental setups in the Lunar Lander environment, each with its unique challenges and objectives.

1) Default Reward Function: Under the default reward function, the SAC algorithm aims to land the spacecraft without any modifications to the intrinsic reward settings of the Lunar Lander environment.

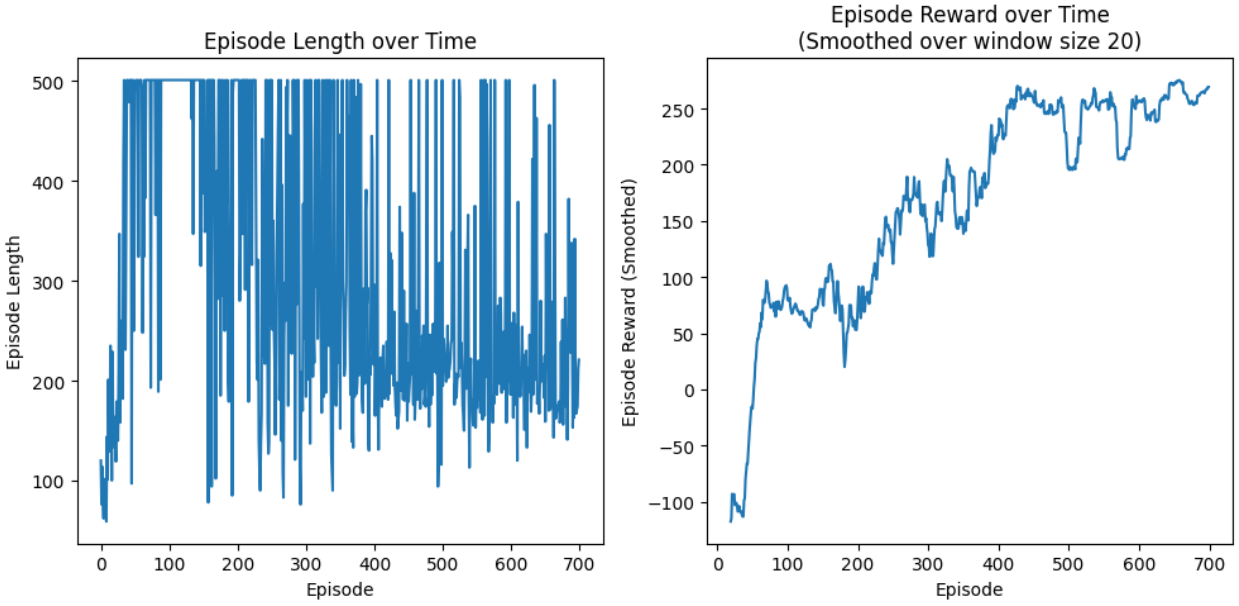

Fig. 12. Training progression of SAC with the default reward function in the Lunar Lander environment.

a) Results and Discussion: Figure 12 shows two plots; the left plot illustrates the length of each episode over time, and the right plot shows the episode reward over time, smoothed over a window size of 20. Notably, the reward trend exhibits a general increase, indicating successful learning. However, the fluctuation in episode length suggests varying complexity in episodes or exploration strategies.

2) Modified Reward Function: In this scenario, the reward function is tweaked to introduce penalties for specific behaviors, such as taking too long to land, and the position aiming to increase the landing speed, enhancing the efficiency of the landing process.

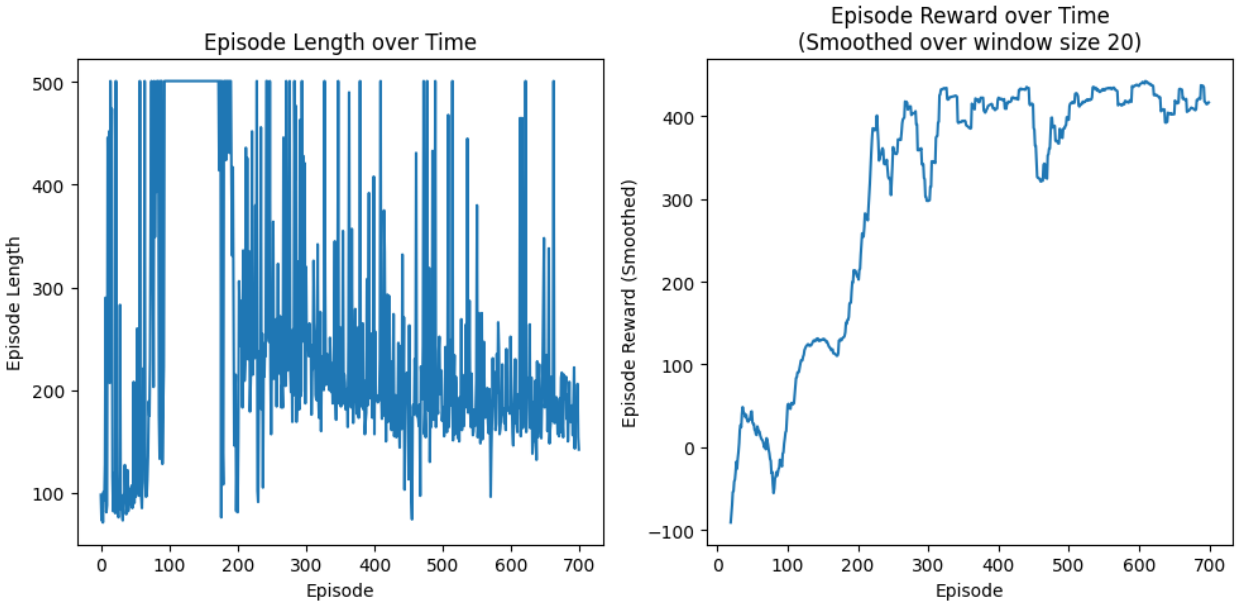

Fig. 13. Effect of the modified reward function on SAC's performance.

a) Results and Discussion: As depicted in Figure 13, the introduction of the modified reward function shows a significant improvement in the reward trend, suggesting that the SAC algorithm has adapted to the constraints and incentives of the new reward structure efficiently. The episode reward shows an upward trend, stabilizing at higher levels than those in the standard environment, demonstrating effective learning and adaptation.

3) Environment with Obstacles: This setup introduces physical obstacles in the Lunar Lander environment, testing SAC's capability to adapt to additional environmental challenges.

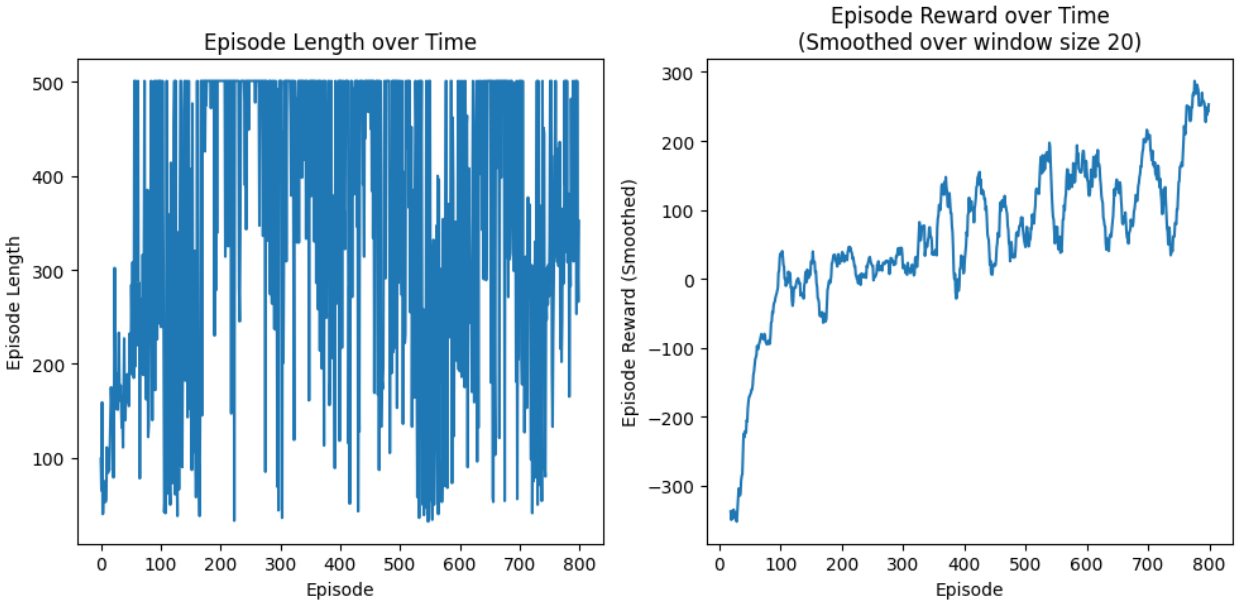

Fig. 14. Training results for SAC in an environment with obstacles.

a) Results and Discussion: Figure 14 illustrates the SAC algorithm's performance under challenging conditions with obstacles. The left plot in the figure shows the episode length over time, indicating considerable variability, which suggests that the agent encounters significant challenges in navigating through obstacles. The right plot of episode rewards, smoothed over a window of 20 episodes, shows a gradual improvement in performance despite initial setbacks, indicating successful adaptation over time. The presence of obstacles likely encourages more complex exploration strategies and requires more robust policy adjustments to achieve similar performance levels as seen in simpler scenarios.

VII Comparison

In the project exploring the Lunar Lander environment from OpenAI Gym, three reinforcement learning (RL) algorithms—Deep Q-Network (DQN), Double DQN, and Soft Actor-Critic (SAC)—were tested across three different scenarios to evaluate their performance under varying conditions. The scenarios included the default environment, a modified rewards system to enhance performance, and an additional obstacle to challenge the navigation capabilities of the agents.

A. Scenario 1: Default Lunar Lander Environment

1) DQN: In the standard setting, DQN showed robust performance by learning to navigate and land the craft effectively. However, DQN's learning curve is relatively slow as it adjusts to the diverse state-action spaces due to its reliance on a single Q-network for value approximation.

2) Double DQN: An extension of DQN, Double DQN, addresses the overestimation bias of the Q-values seen in DQN by decoupling the selection and evaluation of the action in the Q-update step. This results in more stable and reliable learning outcomes. The performance is slightly better than DQN in the standard environment.

3) SAC: As a model-free, off-policy agent using deep learning, SAC's strength lies in its use of a stochastic policy, providing an advantage in exploring the environment effectively. Its continuous action space is particularly beneficial in the default Lunar Lander scenario, leading to smoother landings compared to DQN-based methods.

B. Scenario 2: Modified Reward Function

1) DQN: With enhanced rewards aimed at promoting safer and faster landings, DQN show improved performance in terms of convergence speed, as the clearer reward signals help in learning the optimal policy quicker.

2) Double DQN: Similarly, Double DQN benefits from the modified rewards by refining its estimation of Q-values to optimize for faster and safer outcomes. The reduced overestimation error help in achieving more consistent landings under the new reward conditions.

3) SAC: The advantage of SAC in this scenario could be more pronounced due to its ability to adjust its policy gradient update dynamically, responding effectively to the higher rewards for desired behaviors. Its capacity to balance exploration with exploitation dynamically lead to superior adaptation in scenarios with modified rewards.

C. Scenario 3: Obstacle in Landing Path

1) DQN: Introducing an obstacle presents a significant challenge for DQN. Given its potential difficulty with generalizing from partial observations, DQN preforms relatively good.

2) Double DQN: While Double DQN handle the obstacle slightly better than standard DQN due to its more accurate value estimations.

3) SAC: SAC is the best performer in this scenario due to its inherent flexibility and better handling of continuous state and action spaces. The algorithm's ability to explore different strategies effectively allows it to adapt more quickly to environments where avoiding obstacles is crucial.

D. Conclusion

Overall, each algorithm has its strengths and weaknesses across the different scenarios. DQN and Double DQN are generally more straightforward to implement and can perform well in environments with clear and consistent reward structures but may falter in more complex scenarios involving obstacles due to their reliance on discrete actions and potential overestimation of Q-values. SAC, with its continuous action decisions and robust exploration capabilities, excels in environments requiring fine-tuned control and adaptability, making it particularly suitable for more dynamically challenging scenarios such as those involving obstacles. The choice of algorithm should therefore be aligned with the specific demands and characteristics of the task environment.

VIII Conclusion

This project assessed the performance of three reinforcement learning algorithms—Deep Q-Network (DQN), Double DQN, and Soft Actor-Critic (SAC)—in three scenarios within the OpenAI Gym's Lunar Lander environment. The findings revealed that while DQN and Double DQN performed adequately in standard and modified reward scenarios, they struggled in complex situations involving obstacles. In contrast, SAC excelled across scenarios, showcasing its adaptability and effectiveness in environments requiring precise control and dynamic responses. These results highlight the importance of choosing the right RL algorithm to match the specific challenges of the environment and suggest that advanced methods like SAC are well-suited for tackling complex, real-world problems in reinforcement learning.