Lunar Lander Trajectory Optimization using Reinforcement Learning

By - Vivek Singh 2021UCS1679

About the Problem :

Lunar lander trajectory optimization is a complex challenge in space exploration that focuses on calculating the **most efficient and safe path for a spacecraft to with uncertainty** while landing on the moon crater. Factors such as the Moon's uneven gravitational field, limited fuel resources and the need to avoid hazardous obstacles complicates this task.

Importance of the Project :

It can determine optimal paths [1] that minimize fuel consumption while achieving the desired landing objectives using Reinforcement learning [4]. This efficiency in resource utilization extends mission duration and capabilities.The expertise gained from lunar lander[3] trajectory optimization can be applied to **future space missions and other complex optimization challenges**. ML techniques developed for this project can advance various fields, including robotics, autonomous systems and aerospace engineering.

Challenges:

Modeling complex physics accurately to enable the RL agent to learn precise control strategies for safe lunar landings [6]. Balancing exploration in vast state space to prevent getting stuck in suboptimal solutions during RL training and ensuring learned policies adapt from simulations to the actual lunar environment [3], considering variations and uncertainties.

References :

[1] Soham Gadgil, Yunfeng Xin and Chengzhe Xu, “Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning”,arXiv:2011.11850v1 [cs.LG] 24 Nov 2020.

[2] Toshiki Tanaka, Heidar Malki(Senior Member, IEEE) and Marzia Cescon, “Linear Quadratic Tracking With Reinforcement Learning Based Reference Trajectory Optimization for the Lunar Hopper in Simulated Environment”, Digital Object Identifier 10.1109/ACCESS.2021.3134592

[3] Ashwinikumar Rathod, Ankit Sayane, Chetan Galhat, Hardik Bopche, Chaitanya Shidurkar and Dr. Vinay Keswani, “Rocket Landing Using Reinforcement Learning in Simulation.”, Volume 11, Issue 5 May 2023 | ISSN: 2320-2882

[4] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[5] E. Rodrigues Gomes and R. Kowalczyk, “Dynamic analysis of multiagent q-learning with -greedy exploration,” in Proceedings of the 26th Annual International Conference on Machine Learning, ser. ICML ’09. New York, NY, USA: ACM, 2009, pp. 369–376

[6] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, “Reinforcement learning for robot soccer,” Autonomous Robots, vol. 27, no. 1, pp. 55–73, 2009.

[7] X.-L. Liu, G.-R. Duan, and K.-L. Teo, “Optimal soft landing control for moon lander,” Automatica, vol. 44, no. 4, pp. 1097–1103, 2008.

[8] Y. Lu, M. S. Squillante, and C. W. Wu, “A control-model-based approach for reinforcement learning,” 2019.

[9] C. J. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

<u>Technical Gist Of Reference Papers</u>

● Toshiki Tanaka, Heidar Malki(Senior Member, IEEE) and Marzia Cescon, Linear Quadratic Tracking With Reinforcement Learning Based Reference Trajectory Optimization for the Lunar Hopper in Simulated Environment.

https://ieeexplore.ieee.org/document/9645562

This study is about making a moon-hopping robot avoid obstacles and land safely. The researchers came up with a new plan: first, they designed a smart path for the robot to follow, then they made math rules to control the robot better, and finally, they used a kind of learning to improve both the path and the controls. They tested this on a computer and found it saves fuel and helps the robot land well. But we're not sure if it would work as well in trickier situations that they didn't test.

● Soham Gadgil, Yunfeng Xin and Chengzhe Xu, Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning.

https://arxiv.org/pdf/2011.11850.pdf

This study is about teaching a computer program to play a game called LunarLander. They used two ways to teach it: Sarsa and Deep Q-Learning. The goal was to get high scores in the game. They made the game more difficult to see if the program could still do well. The best scores were around 170+ for Sarsa and 200+ for Deep Q-Learning in the normal game. When the game got even harder, the program still got scores of 100+. They compared the two teaching methods. The study shows that these methods can work for the game, but there might be other ways to teach it or handle really tough parts that they didn't explore.

● Ashwinikumar Rathod, Ankit Sayane, Chetan Galhat, Hardik Bopche, Chaitanya Shidurkar and Dr. Vinay Keswani, Rocket Landing Using Reinforcement

Learning in Simulation.

https://ijcrt.org/papers/IJCRT2305685.pdf

This research is about making a smart system that uses learning to control and land rockets from space. The system learns on its own by trying different things and getting better overtime. They used a special toolkit called Unity MLAgents to build it. The important part is that It helps rockets be used again, instead of being thrown away after one use. But it might only work well in simulations and not consider all the real challenges. They could also talk more about how much computer power this system needs and how it would work for real rockets.

● X.-L. Liu, G.-R. Duan, and K.-L. Teo, “Optimal soft landing control for moon lander,” Automatica, vol. 44, no. 4, pp. 1097–1103, 2008.

Optimal soft landing control for moon lander - ScienceDirect

This paper aims to optimize the control strategy for achieving a soft moon landing with minimal fuel consumption. It transformed into a standard optimal control problem with continuous state inequality constraints.The study introduces the constraint transcription method to convert continuous constraints into canonical form, enabling the development of an efficient computational approach. This method, which combines control parameterization and a time scaling transform, leverages the MISER 3.2 optimal control software package for improved solutions.

DATASETS

The dataset consists of the following components:

- 9,766 realistic renders of rocky lunar landscapes.
- Segmented equivalents of the lunar landscapes, categorized into three classes: sky, smaller rocks, and larger rocks.
- A table of bounding boxes for all larger rocks within the images.
- Processed and cleaned-up ground truth images.

Purpose :
The purpose of this dataset is to provide a sample of artificial yet realistic lunar landscapes, which can be used to train rock detection algorithms. Those trained algorithms can then be tested on actual lunar pictures or other pictures of rocky terrain. This file contains the bounding boxes of all major rocks (colored in blue in ground truth images) that are larger than 10x10 pixels. Each row indicates the frame, the coordinates of the top left corner of the box, and its dimensions. The decimal values for the corner coordinates are given so that the bounding box completely surrounds the rock.

Features :
The dataset contains 5 features :

1. Frame : The considered image number (e.g. 1 = ground0001.tif)

2, TopLeftCornerX : The X coordinate of the top left corner of the bounding box (X = 100.5 means that the rock has no pixel whose X

3, TopLeftCornerY : The Y coordinate of the top left corner of the bounding box (Y = 100.5 means that the rock has no pixel whose Y

4. Length : The X dimension of the bounding box.

5. Height : The Y dimension of the bounding box.

Size :
The dataset contains 9,766 images of rocky lunar landscapes and Number of observations 18867.
The total Size of the dataset is 5 GB.

Source :
We have taken the dataset from the kaggle website :
https://www.kaggle.com/datasets/romainpessia/artificial-lunar-rocky-landscape-dataset

Data Analysis

Three classes are considered: large rocks (in blue), smaller rocks (in green), the sky (in red) and everything else, in black [1].

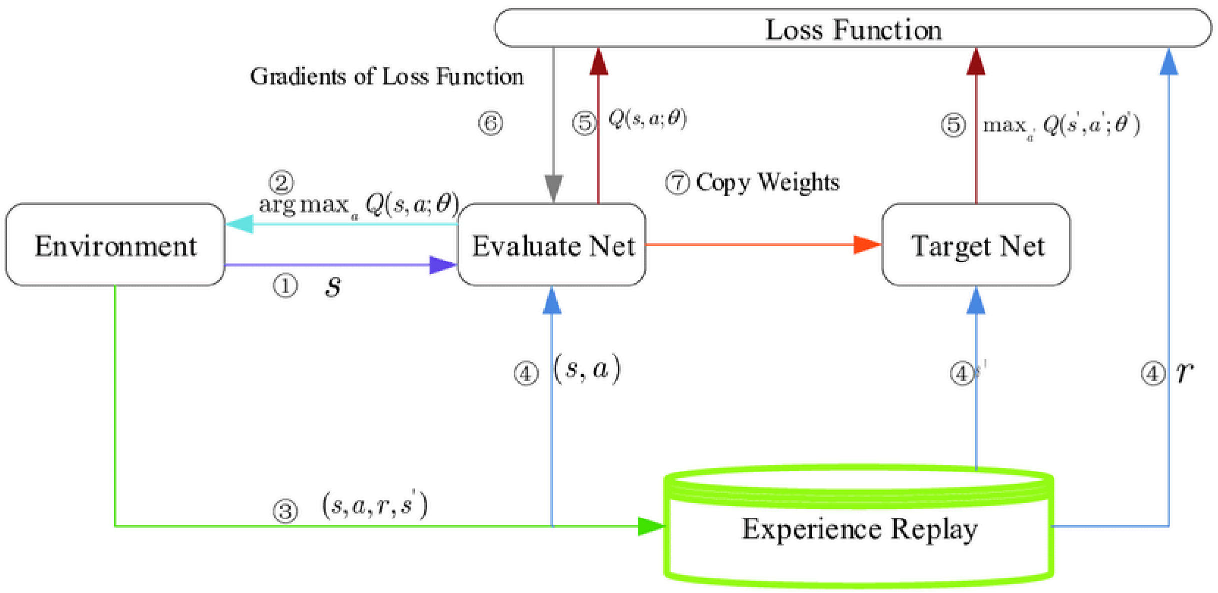
Only rocks [2] measuring more than 10cm are usually represented on the segmented image. This is to avoid cluttering and focus on rocks that are relevant to detect;

Colors go darker for distant rocks. This is also to keep focus on relevant rocks, which are usually closest to the observer. However, the user is free to set their own threshold to determine whether or not a certain rock should be considered depending on its color intensity [2] (cf. the blue rock). As can be seen from the images below, giving a minimum intensity threshold between 50 and 200 is recommended to avoid noise from distant rocks.

Bounding boxes are only drawn around blue rocks of intensity above 150 and dimensions above 20x20 pixels. This is to only consider rocks that are clearly visible while leaving aside those further away.

REFERENCES :
[1] The software used for creating the images and their ground truth is Planetside Software's Terragen.
[2] The authors used the LOLA Elevation Model as a source of large-scale terrain data.

System Architecture



In the lunar lander trajectory optimization using a DQN model, the system comprises several key components. The environment represents the lunar landing scenario, feeding state data to the DQN's evaluate network, which calculates Q-values for actions. Experience replay stores and samples past experiences, while the target network stabilizes the learning process by providing a consistent target. The loss function quantifies the disparity between predicted and target Q-values, guiding the DQN to improve its lunar landing trajectories. Together, these components enable the lunar lander to learn and optimize its descent while interacting with the environment in a reinforcement learning framework.

1. Environment Block:

The environment used for the lunar lander problem is a gym, a toolkit made by OpenAI. This block simulates the lunar landing environment and provides the lunar lander's state information.
There are 8 state variables associated with the state space, as shown below:
state → x coordinate of the lander, y coordinate of the lander, vx the horizontal velocity, vy the vertical velocity, θ the orientation in space, vθ the angular velocity, Left leg touching the ground (Boolean) and Right leg touching the ground (Boolean)
To represent the state of our agent i.e lunar lander at time t: S(t)

2. Evaluate Network Block:

The evaluate network, denoted as Q(s(t), a(t)), is the core neural network in the DQN. It takes the current state s(t) and possible actions a(t) as input and produces Q-values representing expected future rewards for each action.
The Q-value function: Q(s(t), a(t))
We use a 3 layer neural network, with 128 neurons in the hidden layers. We use ReLU activation for the hidden layers and LINEAR activation for the output layer.
The Q-Learning update rule is as follows:
Q(s, a) = Q(s, a) + α(r + γ max a0 Q(s 0 , a0) − Q(s, a)) (1)
The optimal Q-value, Q∗ (s, a) is estimated using the neural network with parameters θ from eqn (1).

Pseudocode for DQN:

```
Initialize network $Q$
Initialize target network $\hat{Q}$
Initialize experience replay memory $D$
Initialize the $Agent$ to interact with the Environment
while $not\ converged$ do
    /* Sample phase
    $\epsilon \leftarrow$ setting new epsilon with $\epsilon$-decay
    Choose an action $a$ from state $s$ using policy $\epsilon$-greedy($Q$)
    $Agent$ takes action $a$, observe reward $r$, and next state $s'$
    Store transition $(s, a, r, s', done)$ in the experience replay memory $D$

    if $enough\ experiences\ in\ D$ then
        /* Learn phase
        Sample a random $minibatch$ of $N$ transitions from $D$
        for $every\ transition\ (s_i, a_i, r_i, s'_i, done_i)\ \ in\ \ minibatch$ do
            if $done_i$ then
                $y_i = r_i$
            else
                $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
            end
        end
        Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2$
        Update $Q$ using the SGD algorithm by minimizing the loss $\mathcal{L}$
        Every $C$ steps, copy weights from $Q$ to $\hat{Q}$
    end
end
```

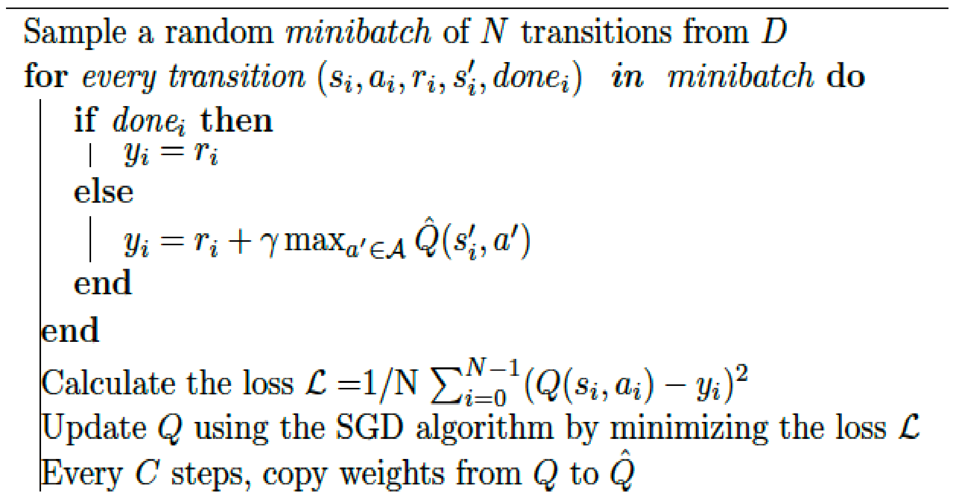
Figure 1

The pseudocode for the evaluate network (Figure 1) outlines the process of calculating Q-values for the given state-action pairs.

3. Experience Replay Block:

Experience replay is a critical component that stores past experiences, {s, a, r, s'}, in a replay buffer and randomly samples mini-batches of experiences during training. This helps in breaking temporal correlations and improving training stability.
The stored experience: {s, a, r, s'} (2)

Pseudocode for experience replay:



```
Sample a random *minibatch* of $N$ transitions from $D$
for *every transition* $(s_i, a_i, r_i, s'_i, done_i)$ *in minibatch* do
    if $done_i$ then
        $y_i = r_i$
    else
        $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
    end
end
Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
Update $Q$ using the SGD algorithm by minimizing the loss $\mathcal{L}$
Every $C$ steps, copy weights from $Q$ to $\hat{Q}$
```

Figure 2

- The pseudocode for experience replay (Figure 2) explains how experiences are stored and sampled for training.

4. Loss Function Block:

The loss function, which calculates the difference between the predicted Q-values from the evaluated network and the target Q-values, is essential for training the DQN. It aims to minimize this difference to improve the lunar landing trajectories.
The optimal Q-value Q∗ (s, a) is estimated using the neural network with parameters θ. This becomes a regression task, so the loss function at iteration i is obtained by the temporal difference error: $L_i(\theta_i) = E_{(s,a) \sim \rho(s,a)} [(y_i - Q(s, a; \theta_i))^2]$ (3)

where $y_i = E_{s' \sim E} [r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})]$ from eqn (1)

Here, θi−1 are the network parameters from the previous iteration, ρ(s, a) is a probability distribution over states s and actions a, and E is the environment the agent interacts with. Gradient descent is then used to optimize the loss function and update the neural network weights. The neural network parameters from the previous iteration,

θ_{i-1}, are kept fixed while optimizing $L_i(\theta_i)$. Since the next action is selected based on the greedy policy, Q-learning is an off policy algorithm.

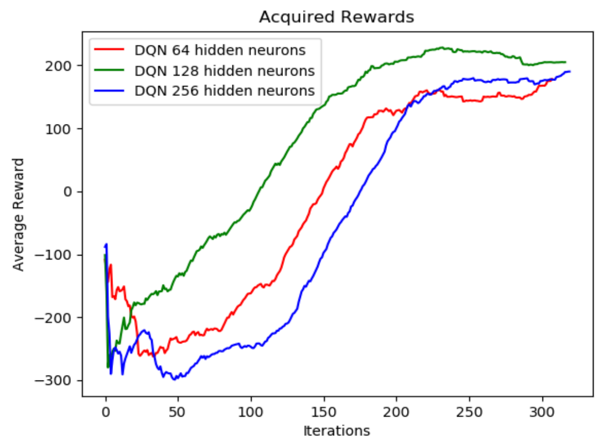
 - The pseudocode for the loss function (Figure 1) describes how the loss is computed and used to update the DQN model.

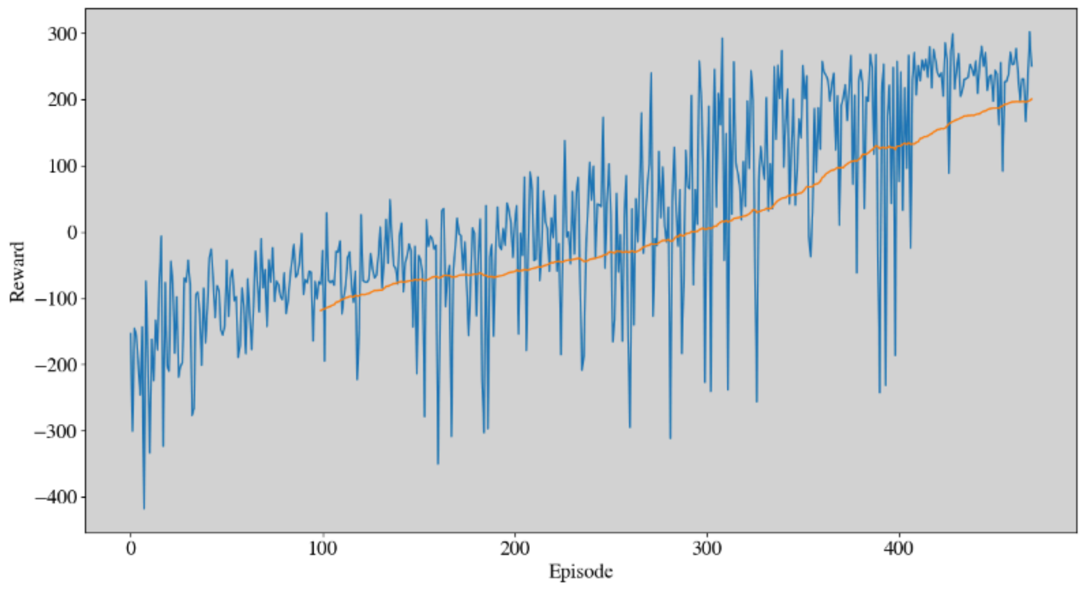
5. Target Network Block:

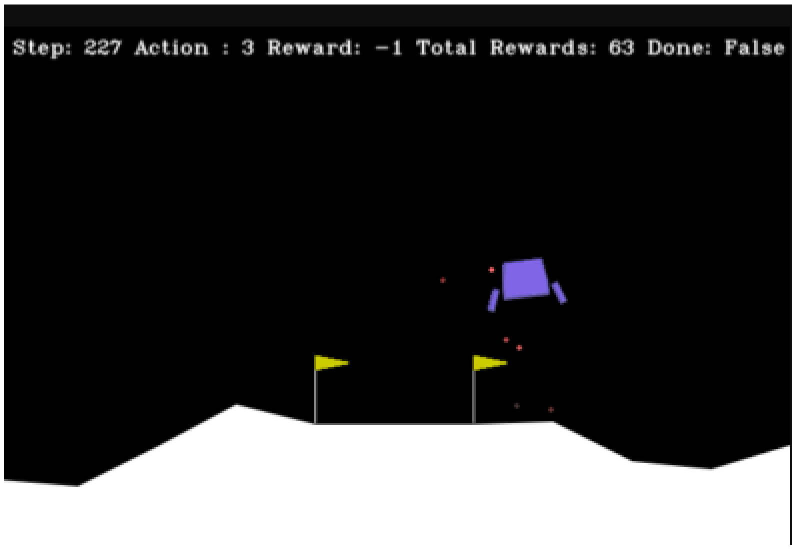
The target network, represented as Q{target}(s(t), a(t)), is a separate neural network used to stabilize the Q-value target during training. It is periodically updated to match the parameters of the evaluated network.
It denotes the target Q-value function: Q{target}(s(t), a(t)).
   - Figure 1 provides pseudocode outlining the periodic updates of the target network using   eqn(1) to align it with the evaluated network.

RESULT :

In the original problem, the DQN agent's average reward, initially negative due to exploration, quickly converges to positive values around 320 iterations, with the neural network having 128 neurons in hidden layers as the chosen hyperparameter. Despite good performance, the DQN agent has a lengthy training time of approximately 2 hours.






Step: 227 Action : 3 Reward: -1 Total Rewards: 63 Done: False

When random forces are introduced, the agent adapts well to small deviations, but struggles with larger forces, resulting in reduced rewards. Noisy observations affect the agent's performance, with a retrained DQN agent under noisy conditions capturing some environmental information but performing less effectively than the vanilla DQN agent.

CONCLUSION :

The DQN agent seems to be erratic.The DQN agent doesn’t discretize the state space and uses all the information that the environment provides. Also, Q-learning is an off policy algorithm in which it learns the optimal policy using the absolute greedy policy by selecting the next action which maximizes the Q-value.There are drops in the acquired average reward which shows that even though the DQN agent is able to achieve higher average rewards, it is not stable.Overall, the DQN agent performance is good.