Using Reinforcement Learning to solve Lunar Lander environment

Valerio Spagnoli 1973484
Matteo Ventali 1985026

June 11, 2025

Contents

1 Introduction

This project aims to address the Lunar Lander environment by applying a reinforcement learning technique called Q-learning. We will explore two different approaches: tabular Q-learning, which is suitable for environments with small and discrete state spaces, and Deep Q-Networks (DQN), a more advanced method that combines Q-learning with deep neural networks, making it capable of handling larger and continuous state spaces.
The goal is to compare these two techniques in terms of learning performance, stability, and efficiency when applied to the Lunar Lander task, which involves controlling a lander to safely reach the ground between two designated flags under the influence of gravity and physics-based dynamics.

2 Basics of Q-Learning

Q-Learning is a reinforcement learning algorithm used to learn the optimal policy

$$\pi^*(s) : S \rightarrow A$$

for an agent interacting with an environment. It is based on learning a Q-function

$$Q(s, a) : S \times A \rightarrow \mathbb{R}$$

which estimates the expected cumulative (discounted) reward of taking an action a in a state s and then following the optimal policy thereafter.

The critical part of the algorithm is how to maintain and update the Q-function. In particular, at every iteration, the algorithm updates its value with one of the following rules:

- **deterministic rule**:

$$Q^{new}(s, a) \leftarrow r + \gamma \cdot \max_{a' \in A} Q(s', a')$$

- **non deterministic rule**:

$$Q^{new}(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in A} Q(s', a') - Q(s, a)]$$

In the previous equations appear these terms:

- α: learning rate;
- r: reward received after taking action a in state s;
- γ: discount factor for future rewards;
- s' and a': respectively, the next state and the next action.

Algorithm 1 Q-learning Algorithm

```
1: Initialize $Q(s, a)$ arbitrarily for all states $s$ and actions $a$
2: for each episode do
3:     Initialize state $s$
4:     while episode not terminated do
5:         Choose action $a$ from $s$ using a strategy
6:         Execute action $a$, observe reward $r$ and next state $s'$
7:         Update $Q(s, a)$:
8:     end while
9: end for
```

The next action strategy must balance:

- **exploration**: taking random actions to explore the environment and gather information;
- **exploitation**: selecting actions according to the policy learned so far to maximize rewards.

2.1 Tabular Q-learning

In the tabular version of Q-learning, the action-value function $Q(s, a)$ is represented explicitly as a table, where each entry corresponds to a specific state-action pair. During training, this table is updated iteratively using the Q-learning update rule. Tabular Q-learning is effective in environments with small, discrete state and action spaces, where it is feasible to store and update the Q-values for all possible combinations. Its main advantages are simplicity, ease of implementation, and theoretical convergence guarantees. However, this approach does not scale to environments with large or continuous state spaces, as the size of the Q-table grows rapidly and becomes impractical. In such cases, approximating the Q-function with a neural network (DQN) or other approaches becomes necessary.

2.2 DQN

In Deep Q-Learning (DQN), the Q-function is approximated using a neural network instead of a lookup table, making it suitable for environments with large or continuous state spaces. Unlike traditional supervised learning, where the dataset is fixed and provided in advance, the training data in DQN is generated incrementally through interaction with the environment. Specifically, during training, the agent builds its dataset in the following way:

- at each time step, the agent observes a transition (s, a, r, s') by interacting with the environment;
- this transition is stored in a memory called *replay buffer*;
- periodically, after some interaction steps, a random mini-batch of transitions is sampled from the buffer to train the Q-network.

To apply the Q-learning update rule, the same network is used both to predict the current $Q(s, a)$ and to estimate the target value $Q^{new}(s, a)$. The power of this approach is the ability of the agent to generalize across similar states and learn effective policies in complex environments.

3 Lunar Lander environment

This environment is a classic rocket trajectory optimization problem. The environment is based on a 2D physics simulation powered by Box2D, and the lander must navigate a gravitational field.

3.1 Observation space

The **observation space** consists of 8 continuous variables representing:

- The lander's x and y positions, ranging from -2.5 to 2.5
- The lander's velocity components, ranging from -10 to 10
- The lander's angular velocity, ranging from -10 to 10
- The lander's angle, ranging from -2π to 2π
- Two binary flags that represent whether each leg is in contact with the ground or not.

The landing pad is always at coordinates $(0, 0)$. Given the continuity of the observation space, a discretization approach was adopted for the implementation of tabular Q-learning.

3.2 Action space

The environment provides a discrete **action space**, composed by 4 main actions, encoded by 4 integers that range from 0 to 3.

- 0: do nothing
- 1: fire left orientation engine (pushing right)
- 2: fire main engine (pushing upward)
- 3: fire right orientation engine (pushing left)

3.3 Reward system

The agent receives a reward based on the quality of the landing:
A successful landing near the center earns positive rewards. Crashes or going out of bounds result in large negative rewards. Fuel usage penalizes the agent slightly,

encouraging efficiency. Leg contact with the ground provides small bonuses. The episode terminates when the lander comes to rest (landed or crashed), making the problem episodic.
A final reward > 200 is considered a success.

4 Experiments

4.1 Lunar Lander with tabular Q-learning

4.1.1 Training results

4.1.2 Running results

4.2 Lunar Lander with DQN

4.2.1 Training results

4.2.2 Running results

4.3 DQN vs Tabular Q-learning

5 Conclusion