Group Project
Training an agent with DQN & DDQN
Lunar Landing & Car Racing

Learning Objectives

In the Group project, you will implement the famous Deep Q-Network (DQN) and its successor Double DQN (DDQN) on an environment of your choice. You can choose between the Lunar Landing and the Car Racing environment defined in the package Gymnasium. The goals of this assignment are to (1) understand how deep reinforcement learning works when interacting with the pixel-level information of an environment and (2) implement a recurrent state to encode and maintain history.

There is a good resource in the internet that can be used as explanation of this practice. See [Rai23].

1 Files for this Practice

```
00_Lunar_lander.ipynb
030_DQN_CARTPOLE_KERAS(empty).ipynb
031_DDQN_CARTPOLE_KERAS(empty).ipynb
```

2 Background on Lunar Lander

This environment is a classic rocket trajectory optimization problem. According to Pontryagin's maximum principle, it is optimal to fire the engine at full throttle or turn it off. This is the reason why this environment has discrete actions: engine on or off.

There are two environment versions: discrete or continuous. The landing pad is always at coordinates (0,0). The coordinates are the first two numbers in the state vector. Landing outside of the landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt.

To observe a heuristic landing, see the notebook

```
00_Lunar_lander.ipynb
```

3 Description of the Group Project

You will implement the Deep Q-Network (DQN) on the game of Lunar Landing using the the Gymnasium environment. The goals of this project are

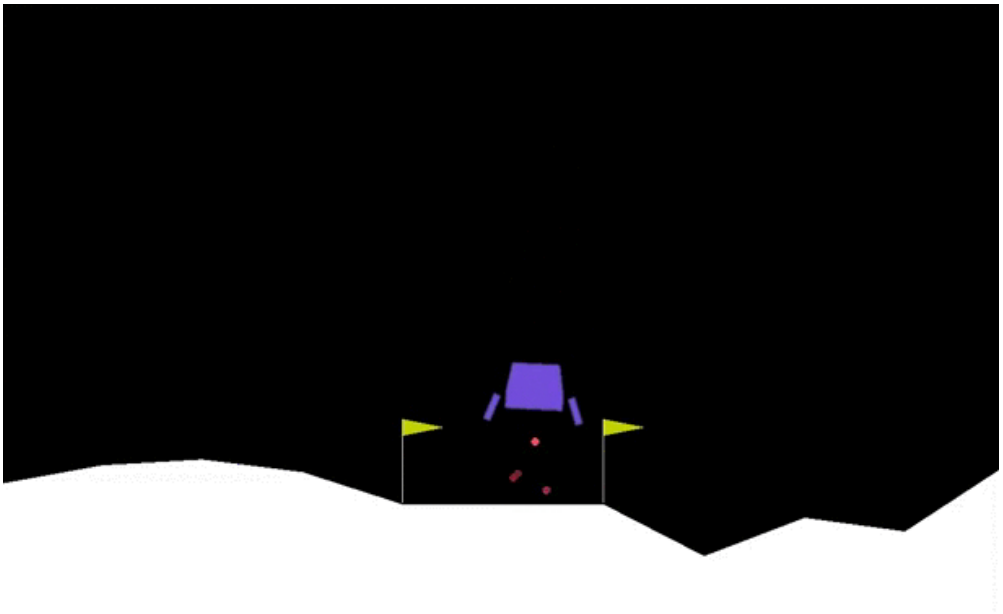

Figure 1: A frame of the lunar lander

- Understand how deep reinforcement learning works when interacting with the pixel-level information of an environment
- Implement a recurrent state to encode and maintain history.
- Use a Deep Learning network for complex agent learning

4 The Lunar Landing environment in Gymnasium

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 & 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.

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. It is considered to be solved if it ends with 200 points. Be careful, negative values happen a lot at the begging.

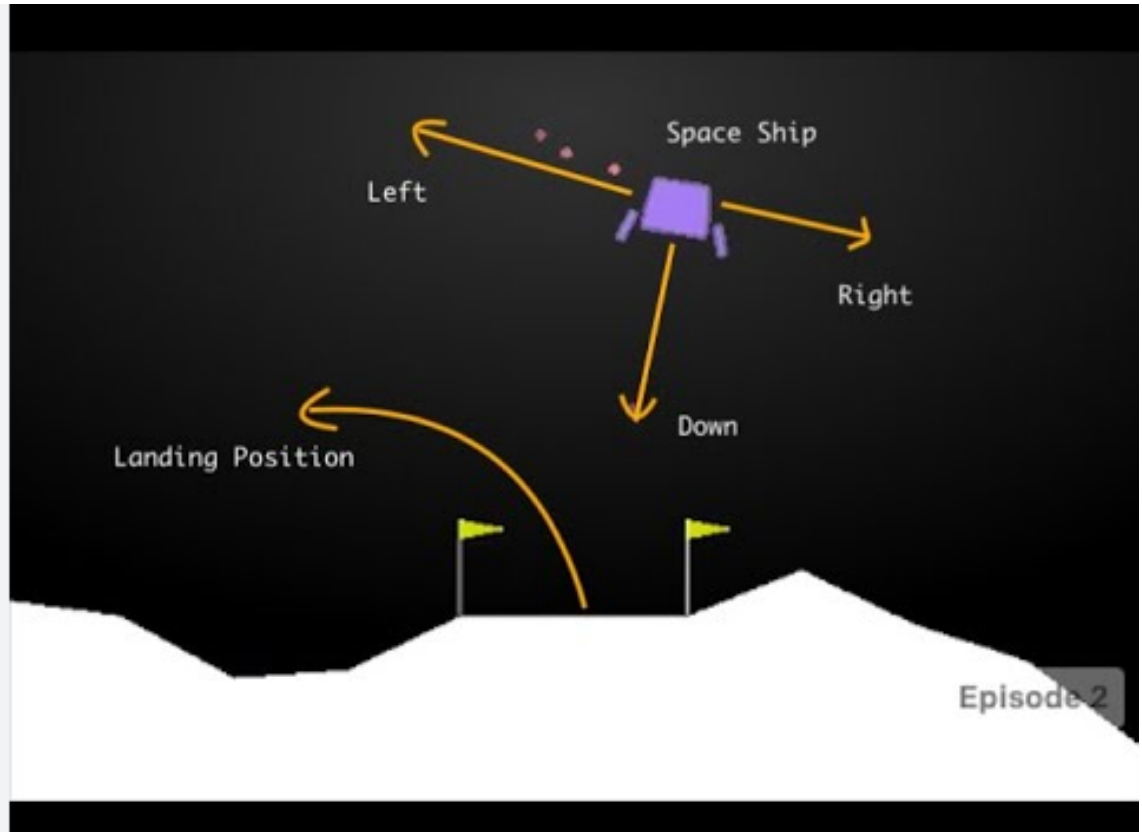

Figure 2: The movements of Lunar Landing

There is a risk with this environment, as the amount of fuel is not controlled, the lunar lander hoovers by using the three engines to stay in the air very long. This is an issue, you will notice that some episodes go forever (up to thousands of steps). For this reason you need to put a limit in the steps on the inner loop of the algorithms. set up a limit on the number of steps. You'll see in the examples that the visualization of the termination is done to see if the lunar lander is starting to hoover.

If you train long enough your lunar lander will learn correctly, but this is a glitch on this environment, maybe to make us think that reality is never perfect and we will find many glitches in real life as well.

Starting State

The lander starts at the top center of the viewport with a random initial force applied to its center of mass. **Episode Termination**

The episode finishes if:

1. the lander crashes (the lander body gets in contact with the moon);
2. the lander gets outside of the viewport (x coordinate is greater than 1);
3. the lander is not awake. From the Box2D docs, a body which is not awake is a body which doesn't move and doesn't collide with any other body:

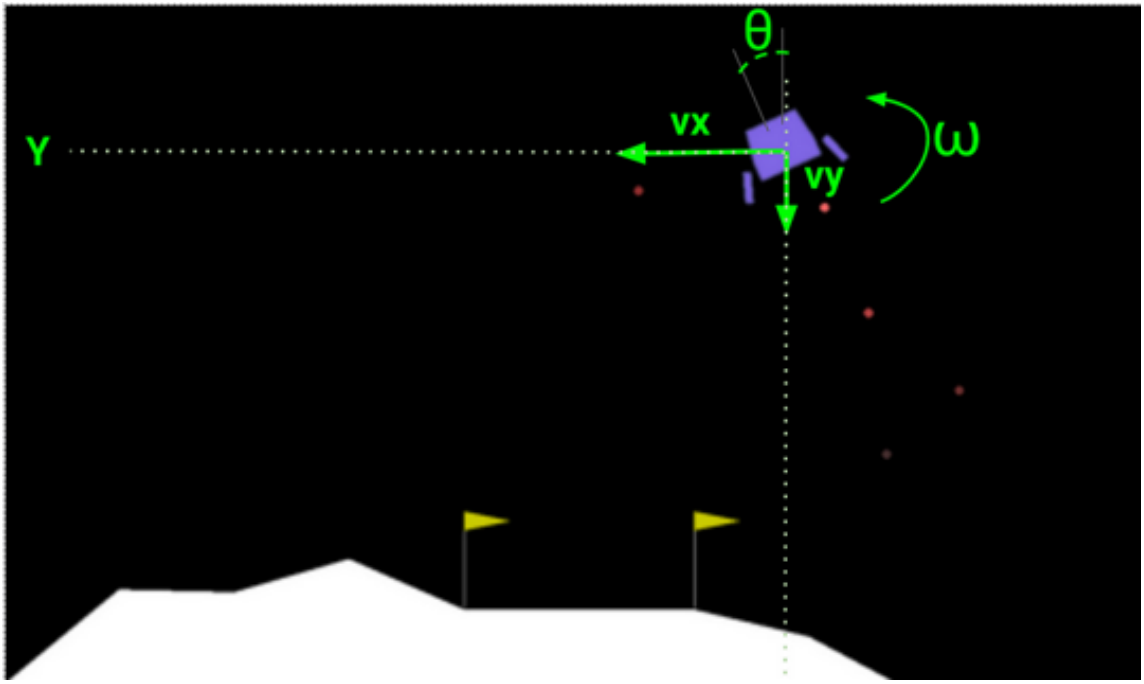

Figure 3: The movements of Lunar Landing

5 The Car Racing environment in Gymnasium

The **Car Racing** environment in Gymnasium is a classic reinforcement learning (RL) environment where an agent controls a car within a 2D track. The goal is to navigate the car, which is initially placed at the starting point, along a winding track while avoiding obstacles and staying on course to complete the race as efficiently as possible.

Key Features

The agent controls a car in a 2D continuous space, with the car's position and orientation determined by various state variables. The track consists of a road surrounded by grass or other obstacles. The car must avoid leaving the road, as doing so results in performance penalties or crashes.

State Space

The state space is continuous, represented by a 96x96 RGB image that captures the car's view of the track. The car's position and velocity are also part of the state, which provides the agent with the necessary context to make decisions.

Action Space

The action space is also continuous, meaning the agent can choose to control the car's acceleration, steering, and possibly other dynamics like braking or gear shifting.

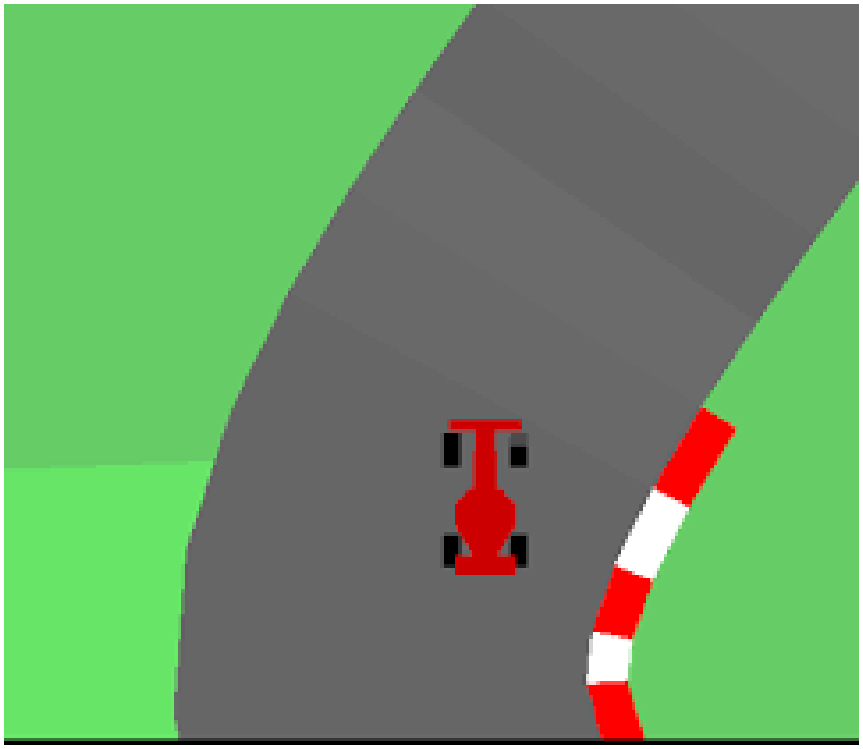

Figure 4: Car Racing

Reward

The agent receives a reward based on its performance in the environment. Positive rewards are typically given for staying on the road and following the track, while negative rewards are given for crashing or deviating too far from the path.

Objective

The agent's goal is to complete the track in the fastest time possible while minimizing mistakes. It needs to learn to balance speed and control to prevent crashes and navigate tight turns effectively.

Observations

The primary observation is an RGB image showing the car's surroundings, which the agent uses to decide how to drive the car in real time.

Use Case

The Car Racing environment is commonly used in RL research to train and test algorithms in a continuous, real-world-like scenario that involves image-based input and continuous control tasks. It's an excellent choice for learning control policies and assessing agent performance in complex environments where the agent must process visual inputs and make decisions based on those inputs.

Overall, the Car Racing environment serves as a great benchmark for training reinforcement learning agents, especially in scenarios that require both fine-grained control and visual processing.

6 Deliverables and submission

You must submit the code of the system. In principle there must be two notebooks

- Training agent notebook (Python)
- Execution of the game with a trained agent (Python)
- Slides from the presentation to the class (Powerpoint, Canvas or similar)

You can add more files or documents if you think are required.

References

[Rai23] Prabhjot Singh Rai. *OpenAI Gym Lunar Lander*. `https://www.psr.ai/projects/lunar-lander`. Accessed: 2024-08-09. 2023.