Question 1

Introduction

Lunar Lander Environment

This environment, provided by Open AI Gym, is a rocket trajectory optimisation problem. The environment is fully observable, single agent, deterministic, episodic, and there are four discrete actions:

- Do nothing
- Fire left orientation engine
- Fire main engine
- Fire right orientation engine

By learning to control these actions, the rocket has to land safely on a landing pad between two flags. After each step, a reward is granted. A reward is increased/decreased depending on a few factors, such as, the distance between the lander and the landing pad, the moving speed of the lander, number of legs contacting the ground, etc. A successfully trained agent needs to achieve a score of at least 200.

Reinforcement learning

To solve this problem, we used Deep Q Network (DQN), a reinforcement learning (RL) algorithm. RL allows an agent to learn through trial and error (without knowing which actions to take), using feedback from its interactions with the environment. RL requires the need to balancing exploration and exploitation, with the aim to maximise the reward. Key components consist of:

- Agent: the decision maker that performs actions.
- Environment: the system where the agent operates.
- State (s): the situation the agent is currently in.
- Action (a): the decisions the agent can make.
- Reward (R): the feedback or result from the environment based on the agent's actions.
- Policy (π): a strategy the agent uses to take actions.

Action-value function

The Action-value function in a state under policy , which gives the expected cumulative reward starting from a state s, taking an action a, is defined as:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s,\ a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s,\ a_t = a\right\}$$

Here, γ is the discount factor that determines the importance of future rewards.

Bellman optimality equation

The Bellman equation provided a recursive decomposition of the value function, relating the value of a state to the values of the following states. Additionally, the Bellman optimality equation is given by:

$$Q(s,a) = r + max_{a'} Q(s', a')$$

It shows that the value of the state-action pair depends on the immediate reward r received after taking action a plus the maximum Q-value for the next states s' over all possible actions a'.

Q-learning algorithm

Furthermore, DQN extends Q-learning by using a neural network approximate the Q-value function, allowing it to handle the environments with large or continuous state spaces. The update rule for Q learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + max_{a'} Q(s', a') - Q(s,a)\big)\ (*)$$

Here, α is the learning rate, controlling how much Q-values are updated during training.

We chose DQN for several reasons:

- Agent can learn optimal landing strategies through trial and error without needing a predefined model of the environment.
- Q-learning handles discretion well, which matches the environment four discrete actions.
- ε-greedy strategies can be used to balance exploration and exploitation.

Theory and Methodology

Deep Q-Networks (DQN)

Traditional Q-Learning uses a table to store Q-values, which is ideal for environments with a small and finite number of states. However, it struggles with large and continuous state spaces, such as the Lunar Lander Environment, due to the size of the Q-table. To overcome this, DQN uses a neural network to approximate the Q-values in unseen states by making reasonable predictions (output) from the observed states (input).

The Q-Network: this is the local network, which is updated continuously and used to select actions during training. The input is the state, and the output is a set of Q-values for all possible actions.
Target Network: a separate neural network that provides more stable targets for the Q-value updates. Unlike the local network, is weights are updated less frequently to prevent rapid oscillation in learning. The update rule is based on the equation (*) shown above.
Experience Replay: to stabilise the learning process, the agent stores its experiences (state, action, reward, next state) in a replay buffer. The network is trained on random mini batches of experiences from this buffer.

Algorithm

To train the agent, we first:

- Initialise replay buffer and networks. Our Q-network has four fully connected layers: the first two have 128 neurons, the third has 64 neurons, and the fourth layer outputs a value for each possible action based on the input state. The target network is a copy of the local network.
- For each step, we select an action using an -greedy policy. If a random sample is greater than the exploration rate (ε), the agent exploits the learned Q-values for the best actions, otherwise, it explores by choosing a random action.
- The experience (action, reward, next state) is stored in the replay buffer.
- The agent then randomly samples mini batches from replay buffer and updates its Q-values by minimising the mean squared error loss.
- Soft update periodically updates the target network to improve stability.

Hyperparameters

The hyperparameters required to achieve successful results are:

- **Learning rate (α)**: controls how much Q-values are updated during training. A high means larger updates, while a lower means a smaller adjustments to the Q-values. It typically ranges from 0.0001 to 0.1.
- **Discount factor (γ)**: is a value between 0 and 1 that determines the importance of future rewards compared to immediate rewards. A higher means the agent prioritises long-term rewards more, while a lower means the agent focuses more on the immediate rewards.
- **Epsilon decay value**: determines how quickly the epsilon value decreases over time. A high decay value allows the agent to explore for a longer period, while a lower value switches to more exploitation sooner.
- **Starting/Ending epsilon value**: sets the initial/final exploration rate at the beginning/end of training. A high value indicates exploration and a low value indicates exploitation.
- **Interpolation parameter:** controls the rate at which the target network is updated with the weights of the local network.
- **Mini batch**: a random sample of experiences from the replay buffer.
- **Replay buffer size**: the maximum number of past experiences stored for training.

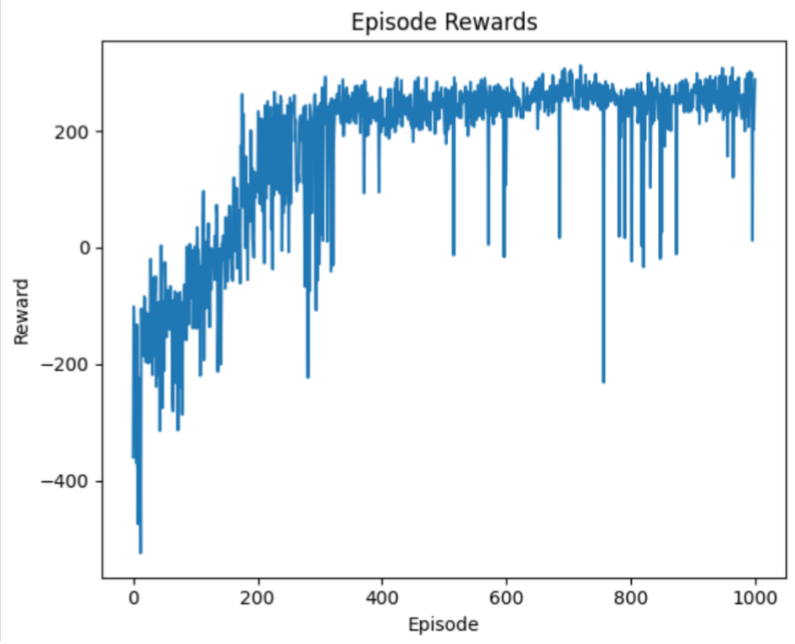
After experimenting, our final hyperparameter values are:

α	γ	ε-decay	Starting ε	Ending ε	interpolation	Mini batch	Buffer size
0.0001	0.99	0.995	0.7	0.05	0.001	150	100000

Results

Reward vs. Episode

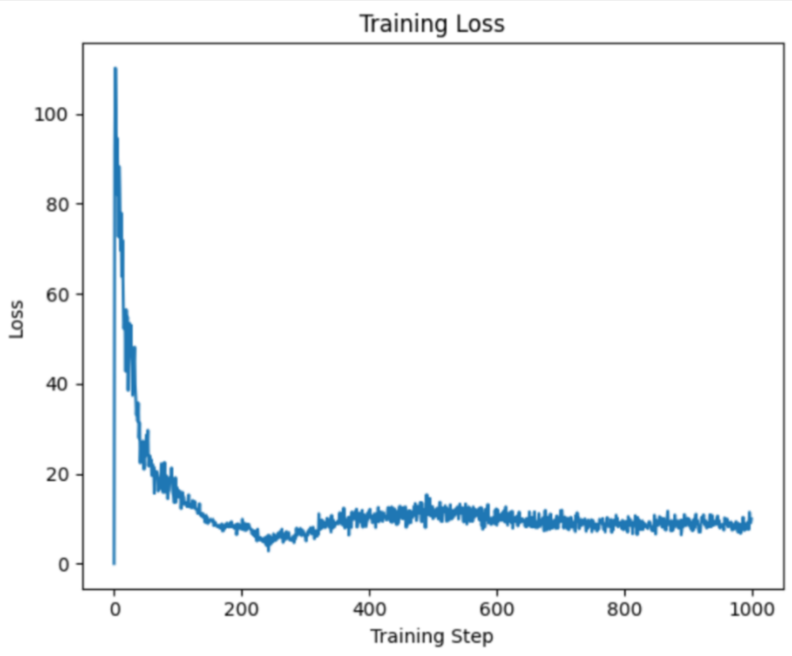
The following plot shows the relationship between the average rewards and episode from 0 to 1000.



Initially, the average score is below negative 200 for the first 50 episodes, and gradually improved and reached positive scores for the next 150 episodes. It plot shows a big jump from episode 200 to episode 250. From here, it gradually increased and oscillated between an average of 200 to 260, with a few dips in between. One of the dips fell below negative 200 at episodes 200+ and 700+, could suggest some failures. Overall, the plot shows stability in training.

Loss vs. Step

The below plot shows the relationship between the training steps (0 to 1000) and average loss values.



There is a rapid decrease in loss values for the initial 100 steps, suggesting that the model quickly learned to reduce errors. After about 200 steps, the loss stabilises at a much lower level compared to the start, with minor fluctuations. This reflects convergence and that the agent effectively learns from its environment while maintaining a balanced update process. No significant spikes indicate smooth training process.

Here are some videos showing the rocket landing on the moon for different episodes.
(To play the videos, please see the ones in the file).

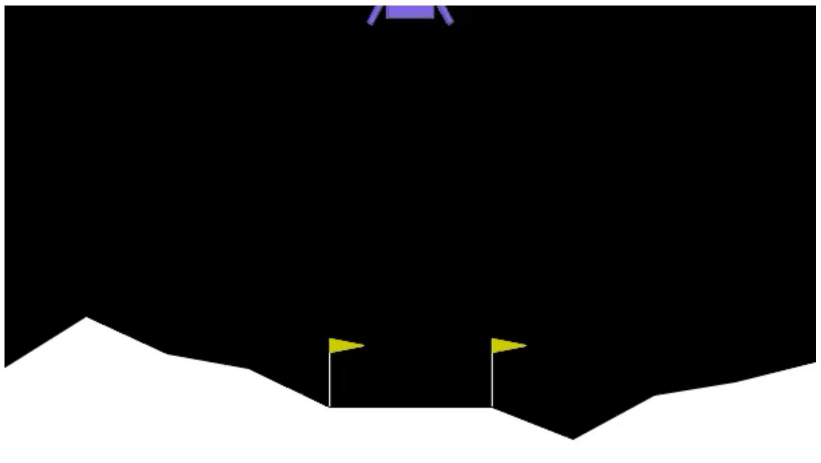


Figure 1: Rocket safely landed between the flags

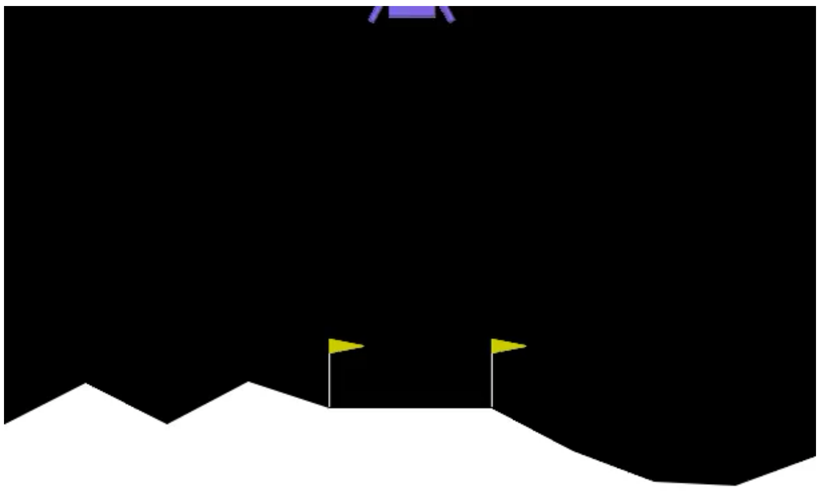


Figure 2: Rocket failed to land between the flags

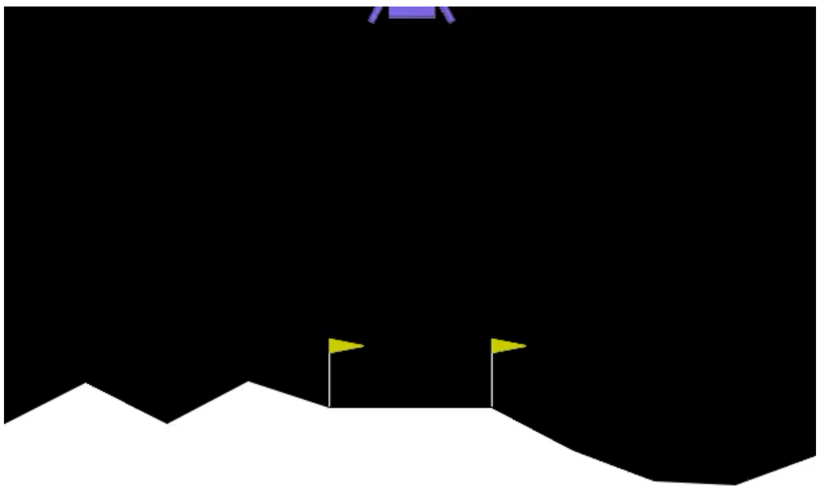


Figure 3: Rocket safely landed outside the flags

Overall, the results show that the DQN performed as expected in the Lunar Lander Environment, demonstrating steady improvement and stability in both rewards and loss. The reward graph shows a clear progression from negative value to consistent positive scores of over 200, with occasional dips. At the same time, the loss graph shows rapid initial learning followed by stable convergence, suggesting effective learning process. The challenges such as occasional dips highlight areas for improvement. This could involve fine-tuning hyperparameters to reduce these dips.

Question 2

The concept of Exploration vs. Exploitation in Deep Reinforcement Learning is fundamentally the same as in Reinforcement Learning (RL):

- Exploitation: using the accumulated information to make decisions that maximise the expected reward.
- Exploration: making decisions with unknown outcomes to discover potentially better rewards.

Too much exploitation may cause the agent to stick to suboptimal actions, while too much exploration may delay optimal performance since the agent is not taking advantage of better rewards. Hence, balancing exploration and exploitation is a challenge in RL. Several strategies have been developed to tackle this. Several approaches that help maintain this balance include:

- **ε -greedy**:
  - With a probability ε, choose a random action (exploration).
  $$a_t = argmax_a Q_t(a)$$
  - With a probability $1 - \varepsilon$, choose the action that maximises the Q-value (exploitation).
  - Decaying ε-greedy: start with a high ε (e.g. 1) and gradually decay it over time (e.g. to 0.1 or 0). This encourages exploration during early training and shifts towards exploitation as the agent learns
- **Softmax Action Selection**: Assigns probabilities to actions using a softmax function based on their Q-values.
  $$P(a) = \frac{e^{Q_t\frac{a}{\tau}}}{\sum_{b=1}^{n} e^{Q_t\frac{b}{\tau}},}$$
  Where τ is ‘computational temperature’, controlling the exploration level. Higher τ leads to more exploration, while lower τ leads to exploitation.
- **Upper Confidence Bounds:** Follows the principle of “optimism in the face of uncertainty”, meaning choosing actions based on a balance of expected reward and uncertainty.

$$a_t = argmax_a(Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}}$$

$N_t(a)$: number of times any action has been taken.

t: timesteps.

c: hyperparameter controlling the exploration weight

The trade-off becomes more nuanced in Deep RL because exploration in high dimensional or continuous state spaces (like in Lunar Lander) can be more challenging, and neural networks can introduce instability during training.

To address these challenges in the Lunar lander problem using DQN, we used:

- ε -greedy with decay to transition from exploration to exploitation.
- Experience replay to stabilise training and improve sample efficiency by learning from past experiences.