Lunar Lander Summary

This was annoying because it basically just crashes into the ground. Taking the idea from the lesson on 11th Feb we should use a DQN network. This is a simple feedforward neural network with two hidden layers of 64 neurons each. It takes the state as input and outputs Q-values for each possible action. The ReLU activation functions help introduce non-linearity, allowing the network to learn complex relationships between states and actions.

1. Neural Network Architecture (DQN class):

```
class DQN(nn.Module):
  def __init__(self, input_size, output_size):
    self.network = nn.Sequential(
      nn.Linear(input_size, 64),
      nn.ReLU(),
      nn.Linear(64, 64),
      nn.ReLU(),
      nn.Linear(64, output_size)
    )
```

Next we need to learn from experience so we set up an Experience Replay Buffer…

2. Experience Replay Buffer:

```
class ReplayBuffer:
  def __init__(self, capacity):
    self.buffer = deque(maxlen=capacity)
```

The replay buffer stores experiences (state, action, reward, next_state, done) and allows the agent to learn from past experiences. This is crucial because:

- It breaks the correlation between consecutive samples

- It allows the agent to reuse important experiences multiple times
- It helps prevent forgetting previous experiences while learning new ones

3. DQL Agent Implementation: The agent uses several key reinforcement learning concepts:

a) Epsilon-greedy exploration:

Ilja explained this briefly but I didn't understand his explanation so I looked it up. Basically imagine you have two choices… you can either always pick something like the best decision seen so far or you can do something randomly different. Imagine you have to pick ice cream. The safe choice is to pick your favourite (maybe vanilla) but it could be that there are other flavours that you would like if only you dared to try them so sometimes you just pick a random flavour and see if you like it. Epsilon greedy allows the model to explore a lot in the beginning and then explore less over time so it starts high (1.0) and always picks a new ice cream and then once it has tried more and more it picks the favourite (highest reward). This helps to avoid a local minimum and maximises the chance of finding the best solution. The decay function has a minimum of 0.1 so it will always explore a bit.

```
def act(self, state):
  if random.random() < self.epsilon:
    return random.randrange(self.action_size)
  # else choose best action…
```

This balances exploration and exploitation:

- Early on, high epsilon means more random actions (exploration)
- As epsilon decays, the agent increasingly chooses actions based on learned Q-values (exploitation)

b) Two networks (policy and target):

```
self.policy_net = DQN(state_size, action_size)
self.target_net = DQN(state_size, action_size)
```

Using two networks helps stabilize training:

- The policy network is updated frequently
- The target network is updated less frequently (every 10 episodes)
- This prevents the Q-values from chasing a moving target

c) Q-learning update:

```
current_q_values = self.policy_net(state).gather(1, action.unsqueeze(1))
next_q_values = self.target_net(next_state).max(1)[0]
target_q_values = reward + (1 - done) * self.gamma * next_q_values
```

This implements the Q-learning update rule:

- Current Q-values are predicted by the policy network
- Target Q-values are calculated using the Bellman equation
- The gamma parameter (0.99) determines how much future rewards are valued

4. Training Loop:

```
def train_agent():
  episodes = 1000
  target_update_frequency = 10
```

The training process:

- Runs for up to 1000 episodes
- Updates the target network every 10 episodes
- Stops early if the agent achieves a reward of 200 or higher
- Uses the gym.wrappers.Monitor to record videos of the agent's performance

The combination of these elements helps the agent learn to:

- Control the lunar lander's thrusters effectively
- Balance fuel consumption with safe landing
- Adapt to different initial conditions

- Learn from both successes and failures through experience replay

The success criteria (reward >= 200) indicates that the agent has learned to land safely and efficiently, considering factors like:

- Landing pad alignment
- Smooth descent
- Minimal fuel usage
- Avoiding crashes

To Do:

Look at prioritised experience replay or double DQN…?