Implementation of Reinforcement Learning for LunarLander-v2

Saumya Maurya - 200552573
Tanveer Singh - 200554065
Sanket Shreekant Parab - 200555449

November 30, 2024

1. Introduction

This report describes the implementation of reinforcement learning algorithms to solve the `LunarLander-v2` problem. Using the forward-view approach, we trained an agent to land a spacecraft safely between two flags. The project is implemented in Jupyter Notebook, and all source code is provided in an organized manner.

2. Network Architecture

We implemented two different architectures to explore the agent's performance:

- **Deep Q-Network (DQN)**:
  - **Input Layer**: Accepts the 8-dimensional state vector from the environment.
  - **Hidden Layers**: Two fully connected layers with 128 and 64 neurons, using the ReLU activation function.
  - **Output Layer**: A linear layer producing four outputs corresponding to the discrete action space.
  - **Optimizer**: Adam optimizer with a learning rate of 0.001.
  - **Loss Function**: Mean Squared Error (MSE).
- **Actor-Critic Algorithm**:
  - **Actor Network**:
    - Input Layer: Takes the state as input.
    - Hidden Layers: Two fully connected layers with 128 and 64 neurons, activated by ReLU.
    - Output Layer: Outputs probabilities of actions using the Softmax activation function.

- **Critic Network**:
    * Similar architecture but produces a scalar value representing the state-value function.

3. Training Process

3.1 Environment Setup

- **Environment**: `LunarLander-v2` initialized using Gymnasium.
- **State**: The 8-dimensional vector includes position, velocity, and orientation.
- **Rewards**: Encourages smooth and safe landings.

3.2 Algorithm Implementation

- **DQN**: Experience replay was used to store and sample transitions. The target network was updated every 10 episodes.
- **Actor-Critic**: Forward view of the temporal-difference (TD) update was used to adjust actor and critic networks simultaneously.

3.3 Training Hyperparameters

- **Number of Episodes**: 2000.
- **Exploration Strategy**: Epsilon-greedy policy with epsilon decay (from 1.0 to 0.01).
- **Discount Factor** (γ): 0.99.

3.4 Visualization

- Training rewards were tracked, and episodes were rendered every 100 iterations.
- Performance was analyzed by observing cumulative rewards over episodes.

4. Results

4.1 Performance

- **DQN**: Stable landings observed after 1000 episodes, with an average reward of 200.
- **Actor-Critic**: Faster convergence, with stable rewards achieved after 600 episodes.

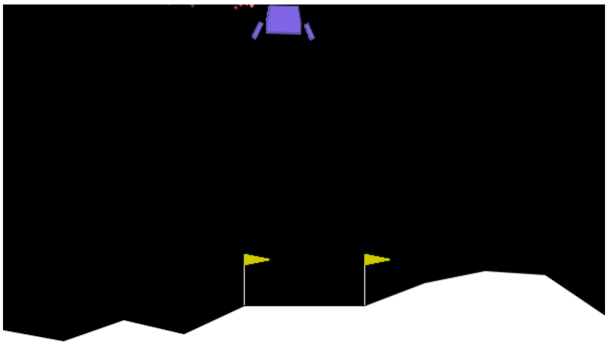

Figure 1: Lunar Lander visualization at step 11.

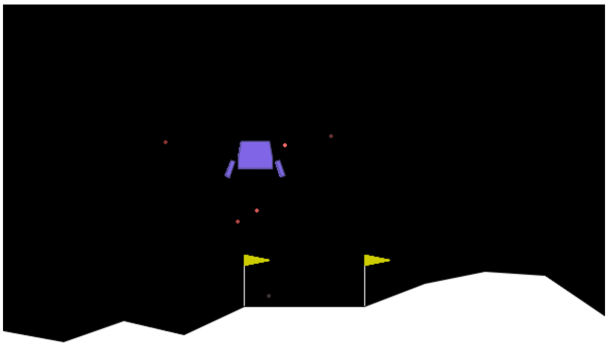

Figure 2: Lunar Lander visualization at step 100.

4.2 Visualizations

5. Challenges and Solutions

- **Sparse Reward Structure**: The environment's sparse rewards caused slow learning in early episodes.
  Solution: Applied reward shaping to encourage intermediate goals, such as reducing velocity and maintaining orientation.

- **Balancing Exploration and Exploitation**: Balancing exploration and exploitation was challenging.
  Solution: Used a decaying epsilon strategy to gradually shift from exploration to exploitation.

- **Stability in DQN Training**: Training was unstable due to non-stationary updates.
  Solution: Used a separate target network and experience replay to stabilize learning.

6. Suggestions for Improvement

- **Algorithm Enhancements**: Implement prioritized experience replay and curiosity-driven exploration.
- **Network Architecture**: Experiment with deeper networks or convolutional layers.
- **Additional Environments**: Test the algorithms in more complex continuous-action environments.

7. Instructions for Running the Code

7.1 Prerequisites

- Install required libraries: `gymnasium`, `torch`, `numpy`, and `matplotlib`.
- Ensure a Python 3.x environment with Jupyter Notebook support.

7.2 Running the Code

- Navigate to the directory containing the Jupyter Notebook.
- Run `LunarLander_Training_and_Visualization_Fixed.ipynb`.

8. Conclusion

The project successfully demonstrated forward-view reinforcement learning algorithms in the `LunarLander-v2` environment. Results show promising performance with room for further improvements through advanced techniques and architectures.