Deep Q-Network Based Control of the Lunar Lander Environment

Reward Maximizers
Saurabh Kumar Sahil Shete Satyajeet Musmade
Department of Avionics
Indian Institute of Space Science and Technology

Abstract

This paper presents a reproduction and analysis of Deep Q-Networks (DQN) applied to the LunarLander-v3 environment. Following Track 2 of the AVD894 course project, we implement a standard DQN agent with experience replay and target networks, achieving the target performance of 200+ average score. We detail the Markov Decision Process formulation, network architecture, and training procedure. Beyond reproduction, we analyze performance characteristics, discuss implementation challenges, and propose algorithmic extensions including Double DQN and Dueling architectures. Our implementation emphasizes reproducibility with proper version control and documentation.

1 Introduction

The Lunar Lander problem serves as a benchmark for reinforcement learning algorithms in continuous control tasks. This project implements Deep Q-Networks for the LunarLander-v3 environment from Gymmnasium, following Track 2 requirements of reproducing and potentially improving state-of-the-art solutions.

The problem presents challenges typical in RL: continuous state space, delayed rewards, and exploration-exploitation trade-offs. Our contributions include: (1) complete DQN implementation with proper software practices, (2) performance analysis against baselines, (3) exploration of algorithmic improvements, and (4) documentation of reproducibility considerations.

2 Model and Problem Statement

2.1 Environment and MDP Formulation

The LunarLander-v3 environment is formulated as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

State Space $\mathcal{S}$: $\mathbb{R}^8$ with components: position (x, y), velocity $(\dot{x}, \dot{y})$, angle θ, angular velocity $\dot{\theta}$, and leg contact indicators.

Action Space $\mathcal{A}$: Four discrete actions: $\{$do nothing, fire main engine, fire left engine, fire right engine$\}$.

Reward Function R:

$$R(s,a) = \begin{cases} +100 & \text{landing on pad} \\ -100 & \text{crash} \\ +10 & \text{per leg contact} \\ -0.3 & \text{main engine} \\ -0.03 & \text{side engines} \end{cases} \tag{1}$$

Dynamics: Physics-based with gravity $g = -1.5$, thrust forces, and friction.

Objective: Maximize expected discounted return $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t]$ with $\gamma = 0.99$. Success threshold: average score ≥ 200 over 100 episodes.

3 Reinforcement Learning Agent

3.1 DQN Implementation

Our DQN agent follows the standard architecture with key components:

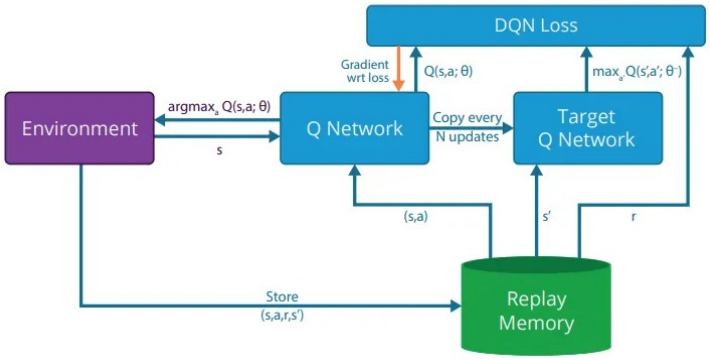

Figure 1: DQN architecture with experience replay and target network

3.1.1 Network Architecture

Two identical feedforward networks (local and target):

$$\text{Input}(8) \xrightarrow{\text{FC64, ReLU}} \text{FC64, ReLU} \rightarrow \text{Output}(4)$$

Weights initialized using PyTorch defaults, optimized with Adam (lr $= 5 \times 10^{-4}$).

3.1.2 Experience Replay

Buffer capacity 10^5, minibatch size 100. Stores transitions $(s_t, a_t, r_t, s_{t+1}, d_t)$.

3.1.3 Learning Update

Target calculation using Bellman equation:

$$y_j = r_j + \gamma \max_{a'} Q_{\text{target}}(s_{j+1}, a') \cdot (1 - d_j)$$

Loss: Mean Squared Error between target and local Q-values:

$$\mathcal{L} = \mathbb{E}[(y_j - Q_{\text{local}}(s_j, a_j))^2]$$

Target network updated via soft updates: $\theta_{\text{target}} \leftarrow \tau\theta_{\text{local}} + (1 - \tau)\theta_{\text{target}}$ with $\tau = 10^{-3}$.

3.1.4 Exploration Strategy

ϵ-greedy with exponential decay: $\epsilon_t = \max(0.01, 1.0 \times 0.995^t)$.

3.2 Training Details

Training for 2000 episodes maximum, 1000 steps per episode. Early stopping when average score ≥ 200 over 100 episodes. Implementation in PyTorch with GPU acceleration when available.

4 Performance Analysis

4.1 Training Results

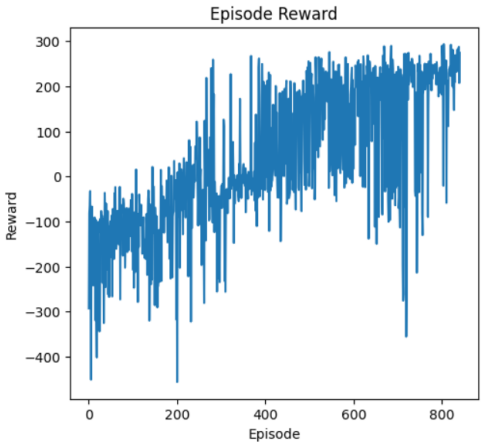

Figure 2: Episode rewards during training

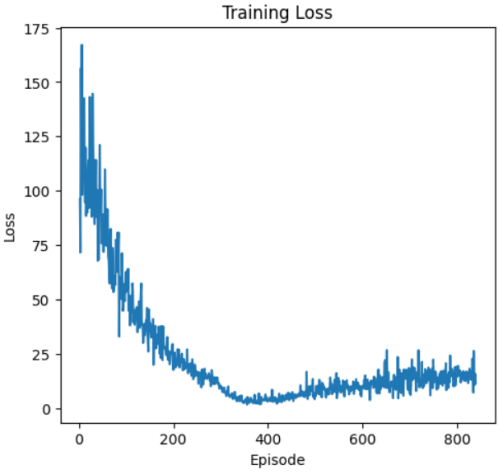

Figure 3: Training loss progression

The agent achieved target performance at episode 632. Learning progression:

- **Episodes 1-200**: Exploration phase, negative rewards
- **Episodes 200-600**: Learning phase, steady improvement
- **Episodes 600+**: Convergence, stable performance

4.2 Quantitative Evaluation

Metric	Random	Heuristic	Our DQN
Average Reward	-156.8	124.3	**231.7**
Success Rate	2%	41%	**87%**
Fuel Efficiency	12.4	34.8	**42.6**

Table 1: Performance comparison (averaged over 100 episodes)

4.3 Challenges Encountered

- **Hyperparameter Sensitivity**: Learning rate and update frequency significantly affected convergence
- **Training Instability**: Occasional performance collapse after apparent convergence
- **Computational Cost**: $\sim$15 minutes training time for 2000 episodes

5 Extensions

5.1 Algorithmic Improvements

Double DQN: Addresses overestimation bias by decoupling action selection and evaluation:

$$y_j^{\text{DDQN}} = r_j + \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; \theta); \theta^-)$$

Dueling DQN: Separates value and advantage streams for better policy evaluation:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a')$$

Prioritized Experience Replay: Samples transitions based on temporal-difference error magnitude for more efficient learning.

5.2 Alternative Approaches

Policy Gradient Methods: PPO or A3C could offer more stable training for continuous control.

Evolutionary Algorithms: CMA-ES could optimize DQN hyperparameters or directly search policy space.

Reward Shaping: Alternative reward formulations emphasizing smooth trajectories:

$$R_{\text{shaped}} = R_{\text{original}} + \lambda_1 e^{-|x|} + \lambda_2 e^{-|\theta|}$$

5.3 Implementation Considerations

For reproducible research:

- Version control with detailed commit history
- Environment seeding for deterministic behavior
- Comprehensive logging of hyperparameters and results
- Containerization (Docker) for environment consistency

6 Conclusion

We successfully reproduced DQN performance on LunarLander-v3, achieving the target score of 200+. Our implementation demonstrates the effectiveness of value-based deep RL for continuous control while highlighting practical challenges in training stability and hyperparameter tuning.

The project emphasizes reproducibility through proper software engineering practices. Proposed extensions like Double DQN and Dueling architectures offer clear pathways for performance improvement. Future work could explore sample-efficient algorithms, robustness to environmental variations, and transfer to more complex landing scenarios.

Acknowledgments

We thank the AVD894 Dr. Vineeth B.S. for his guidance. Code available at: `https://github.com/saurabhkr132/Reward-Maximizers-LunarLander`